ライブラリ
C 103, 105
ラベル
domainname 57

リージョン 41
リピーター 9, 12

ルーター 12

レコード 96

メールアドレスの別名 61

簡単な会話 71

グラフ
LaTeX に取り込む 131
関数の 128
離散データの 129

計算環境 1

ゲートウェイ 12
言語のモード
emacs の C モード 108

サブドメイン 57

シェル 19
書式つき 97
書式なし 97

実行モード 28, 31

スクロールバー 48

ソースリスト
C 106

タイトルバー 53
他のユーザに関する情報 69

地物標準環境 14, 17

電子メール 58
電子メールアドレス 58

トランシーバー 9
トランシーバーケーブル 9

ドメインネームシステム 56
ドラッグ 49, 50

内部表現 89

日本語ローマ字入力モード 42
ニュースグループ 61

ハッシュ 31

バックグラウンド 32, 38
バッファ
emacs の— 38

光トランシーバー 12
光ファイバー 9
左タイトルボタン 53
標準環境 14

ファイル形式 33
ファイル転送 68
不定 80

物理メール 58
ブリッジ 12
ブロック 97
分割コンパイル
C 106

プリンター 28
TeX から 122
GNUPLOT から 130
プレビュー
TeX の— 122
プロトコル 9

ヘルプ
GNUPLOT の— 133
変数
シェルの— 31

ページプリンター 115

ポイント
emacs の— 38

マーク
emacs での— 41
マウスカーソル 47
マシン名 57
マニュアル
GNUPLOT の— 133
マルチポートトランシーバー 9

ミニバッファ 38

メールの転送 60

モード
emacs の— 38
モードライン 38

ASCII 版 120
LaTeX
GNUPLOT から 131
LaTeX コマンド 122
LESS 37
less 35
LESSCHARSET 37
lint 106
login 15
logout 16
lower 50
lpq 28
lpr 28, 115
lprm 28

make 106
Makefile 108
man 33
MANPATH 35
mh-e 59
mh-profile 59
MORE 37
more 35, 37
move 50, 53

network news 61
nkf 29

PAGER 37
pager 35
path 28, 30
plot
GNUPLOT の— 128
PostScript 28, 115
GNUPLOT から 130, 133
ps 25

queue 28

raise 50
raise-lower 53
RARP 9
rehash 28, 31
resize 50, 54
root window 17, 18, 49

SEQUENTIAL 97
sh 19
SHELL 37
shell 19, 29

TCP 9
TCP/IP 9
tcsh 19, 32
telnet 9
TeX
ASCII 版 120
twm 47

vi 23

window manager 47

X 17
X-window 17, 45
xcalc 52
xcolors 52
xdvi 122
xfontsel 52
xmag 52
xman 52
xrefresh 52

yank 41

イーサーネット 9
印刷
TeX の— 122
GNUPLOT の— 130

ウィンドウ 45
emacs の— 38
ウインドウマネージャー 47

エコー領域 38
演算子
GNUPLOT の— 128

オンラインヘルプ
GNUPLOT の— 133

カーソル
emacs の— 38

索 引

.cshrc 32
.forward 60
.login 32

anonymous ftp 69
ANSI 規格 103, 104
ARP 9
auto-save 27

back ground process 32, 38
back-up 27
boiled-egg 43

C 言語 103
C-shell 19, 32
cat 35
cc 103
chfn 70
COMMON 88
core 27
csh 19, 32
C のライブラリ 33
C 言語 99

De-iconify 50, 53
DIRECT 97

EDITOR 37
eepic 131
EGG 37
egg 43
emacs 21, 37, 52
ENSCRIPT
    環境変数 28
enscript 29
ENTRY 82
epic 131
epoch 52
epsf
    GNUPLOT から 133
euc 29
EXTERNAL 87

FAQ 26
finger 69
focus 50
foo 28
FORTRAN 78
FORTRAN66 78
FORTRAN77 78
ftp 9, 68

gcc 103, 104
GECOS 70
geoph 14, 15, 58
GNUPLOT 128
gnus 62

hash 31

icon 50, 53
Icon Manager 48
iconify 50, 53
info 35, 37
Internet 9
INTRINSIC 87
IP 9
IP アドレス 9

jtops 29

kill
    命令 25
kterm 18, 47, 52

LAN 9
LaTeX

References

アドビ ・システムズ, 1989: PostScript チュートリアル & クックブック, アスキー出版局.

上原政二, 1988: 異機種接続と LAN 絵とき読本, オーム社.

浦昭二, 1982: FORTRAN 77 入門, 培風館.

大野義夫 ・編, 1989: TEX 入門, 共立出版.

奥村晴彦, 1991: LATEX 美文書作成入門, 技術評論社.

カーニハン B. W., プローガー P. J., 1982: プログラム書法 第 2 版, 共立出版.

カーニハン B. W., リッチー D. M., 1981: プログラミング言語 C — UNIX 流プログラム書法と作法—, 共立出版.

カーニハン B. W., リッチー D. M., 1989: プログラミング言語 C 第 2 版 — ANSI 規格準拠 —, 共立出版.

釜江, 1989: 東京大学理学部国際理学ネットワークについて, 理学部広報, **21-2**.

クヌース D. E., 1989: TEX ブック, アスキー出版局.

ストールマン R., 1986: GNU Emacs マニュアル, 共立出版.

西村恕彦, 酒井俊夫, 高田正之, 1986: 岩波 FORTRAN 辞典, 岩波書店.

林晴比古, 1991: 新 C 言語入門 — 入門編, シニア編, 応用編, スーパービギナー編—, ソフトバンク株式会社出版事業部.

ランポート L., 1990: 文書処理システム LATEX, アスキー出版局.

第 8 章

著作権

Copyright © 1992, 地球惑星物理学科地球惑星物理学演習テキスト編集グループ

本テキストに関する著作権は、地球惑星物理学科地球惑星物理学演習テキスト編集グループ (以下 「グループ」と略す) に属します.

本テキストは 「グループ」に無断で複写・複製することを禁止します. また, 本テキストのソースプログラム (LaTeX とポストスクリプトファイル) も無断で再配布・変更してはいけません. 本テキストの一部もしくは全部を引用する場合は, 当 「グループ」による 「地球惑星物理学演習テキスト」からであることを明記して下さい.

なお、著作者 「地球惑星物理学科地球惑星物理演習テキスト編集グループ」の連絡先は

ta_comp@geoph.s.u-tokyo.ac.jp

です. 連絡はすべて電子メールでお願いします.

- 3 章に含まれていた 「Suntools」の節は省き 「X-window」のみにしました. また, 「X-window」の節も地物標準環境に合わせて大幅に書き換えました.
- 4 章に含まれていた 「ecc での bbs の使い方」の節は省きました.
- 5 章に含まれていた Lisp, Miranda, 「その他の言語」の節を省き, 新たに書いた 「最低限の FORTRAN 入門」, 「最低限の C 入門」, 「FORTRAN と C の比較」, 「良いプログラムの書き方」の 4 個の節を入れました. また, 元の 「C 言語」の節は 「少し進んだ C 言語の話題」として残しました.

この他にも, 非常に小さな変更が少しあります.

「初めて使う人のための S Workstation ガイド」(2.2 節) については次のような変更を加えました.

- 「はじめに」, 「基本的な注意事項」, 「他にどんなことが出来るのか」, 「次の段階のガイド」, 「さいごに」の節は省略しました.
- 「無保証の宣言」も省略しました.
- 「エディタ emacs」の節に, 基本的なキー操作のまとめを付け加えました.
- シェルのプロンプト
- 「chfn — 本名の登録」の学科と名前

「FORTRAN77 規格について」(5.2 節) および 「コードの内部表現と FORTRAN」(5.3 節) は 「地球流体電脳倶楽部」から提供していただいた地球流体電脳ライブラリマニュアル MATH1, MISC1 をもとにしています.

また, 東京大学 工学系研究科 情報工学専攻の毛利隆夫氏, 慶應義塾大学 環境情報学部の永井清登氏, その他の皆様にも, 執筆の参考になる資料を提供したり紹介したりして頂きました. 感謝致します.

その他

テキスト作成に先立ち, さまざまな参考資料を提供していただいた地球惑星物理学科専攻課程の教官方に感謝致します. また, 気候システム研究センターの中島健介, 山中康裕両氏, ならびに, 東京都立大学の増田耕一氏には, テキスト作成を通じて多大なる助言をいただきました.

第 7 章

謝辞

2 章 − 6 章

2 章から 6 章までは, 本来, 基本的には,「教育用計算機センターの Unix Work Station の使い方 (Ver 1.0)」(編著者: 下國 治) という, 工学部応用物理系の有志が自主的に書いた冊子でした.

この冊子の表紙の裏側には, 次のような著作権などに関する宣言が書かれています:

> Copyright ©April,1992. Osamu Shimokuni
>
> 著作権表示とこの許可告示が全ての写しに載せられている場合にのみ編著者である下國 治は、いかなる人間に対しても、このドキュメントの配布を自由に行なうことを許可します。
>
> また、本文の内容の改変については、改変者の氏名を告示し、改変した場合に生じた不具合についての責任は改変者が負うものとし、編著者は一切責任を負う義務を持たないという条件下で、上記の編著者に断ることなく何人も自由に行なうことを許可します。
>
> このドキュメントを読んだことによって生じるいかなる損害についても編著者はその責任を負う義務を持ちません。これはこのような損失が起こる可能性について読者が知らされていた場合でも、あるいは如何なる団体からの申したてでも例外ではありません。

また, 2.2 節は,「初めて使う人のための S Workstaion ガイド」(物理工学科 4 年, 田丸 博晴, 平成 4 年 5 月 22 日) という冊子でした.

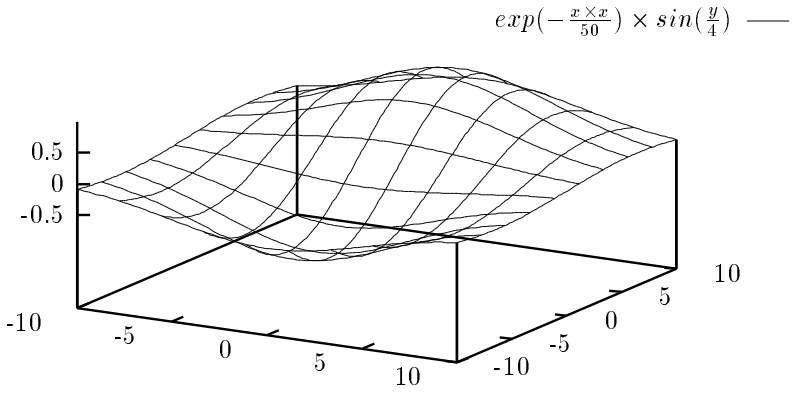
田丸さんからは, GNU GPL[1] の条件下での再配布の許可を頂いています.

以上の 2 冊の冊子の LaTeX 原稿を下さった下國 治 氏を始めとして, 応物系有志の方々に, 深く感謝します.

「教育用計算機センターの Unix Work Station の使い方」(2 章から 6 章まで) の大部分は元の原稿のままですが, 地球惑星物理学科地球惑星物理学演習テキスト編集グループは, 次のような変更を加えました.

- Dedication,「はじめに」, 執筆者一覧, 謝辞は省きました.
- 上に述べたように, 2 章には, 本来「初めて使う人のための S Workstation ガイド」であったものを,「取り敢えず使うために」(2.2 節) として加えました.
- 全体にわたって応物系に特有な内容は, 地球惑星物理に適するように書き換えました. それにともない,「応物標準環境」の代わりに「地物標準環境」を作り, 2.1「地物標準環境」の節も新たに付け加えました.

[1] "The GNU General Public License" の略。詳細は GNU Emacs に附属しているドキュメントや「GNU Emacs マニュアル」(ストールマン, 1986) に任せますが, 要約すると: 無料, 無保証, 自由に利用, 他人に譲れるが, 著作権は放棄しない; 商業的利用を禁ずる,... という条件です.



epsf を使う方法

epsf とは encapsuled postscript file の略で, これはプリンタに打ち出す時に使った, junk.ps などのことです. この方法を使う利点は, 縦軸の見出しが９０度回転して出力されるので, 長い見出しのときに読みやすいということです.

もうすでに, junk.ps というファイルは作ってあるとします. これを, test.tex の中に取り込みたいとします. まず, test.tex の冒頭で

```
\documentstyle[a4j,epsf]{jarticle}
```

のように, epsf スタイルファイルを取り込みます. そして文章中の取り込みたい位置で,

```
\begin{center}
\epsfile{file=junk.ps,height=15cm,width=10cm}
\end{center}
```

というようにして取り込めます. 実際に取り込まれるのは, dvi2ps コマンド[8]を使ったときなので, test.dvi だけでは, 印刷できません.

6.4.6 さらに詳しい説明について

以上簡単に GNUPLOT の説明をしてみました. さらに詳しい説明は, GNUPLOT のオンラインマニュアルに書かれています. GNUPLOT をたちあげて,

```
help
```

と入力すると概要説明が始まり, 次々に項目を選択しながら説明を読むことができます.

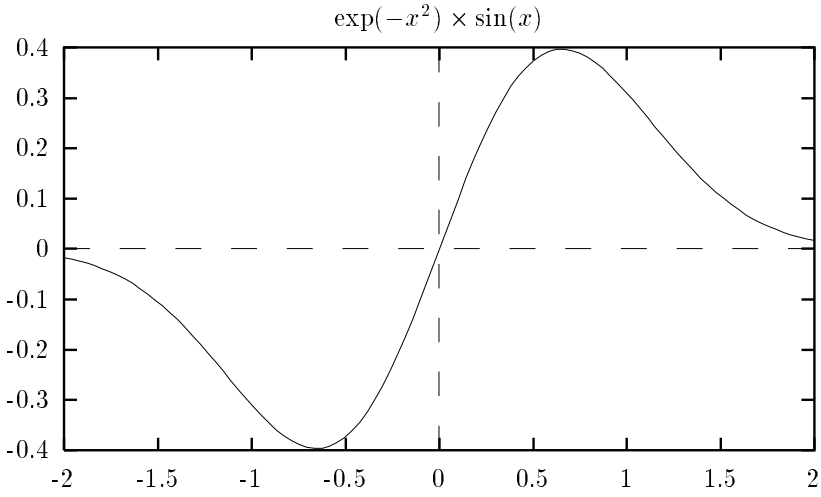
[8] geoph のプリンタは PostScript プリンタなのでこれを使って, .dvi ファイルを PostScript 形式に変換します

```
set terminal eepic
set output 'gpgraph.tex'
set title '$\exp(-x^2)\times\sin(x)$'
set xrange[-2:2]
set nokey
plot exp(-x*x)*sin(x)
set output
exit
```

このようにして GNUPLOT で作ったファイルを

```
\begin{center}
\input{gpgraph.tex}
\end{center}
```

このようにして取り込むと次のようになります.



もう一つ

```
set terminal eepic
set output 'gpgraph2.tex'
splot exp(-x*x/50)*sin(y/4)
set output
exit
```

```
lprm 594
```

というようにすると印刷命令が取り消されるはずです. もう一度 lpq を使って, 本当に取り消されたかどうか確認してください. 自分が指定した印刷を全て取り止めるときは,

```
lprm s22516
```

のように自分の ID を指定します.

6.4.5 グラフを LaTeX の文章中に取り込む

ここでは, 2 つの方法を軽く説明します. TeX 自身の使い方は, ここでは説明しません.

eepic を使う方法

eepic とは extentions to epic の略で, epic とは enhancements to picture envirionment の略です. この方法を使う利点は, グラフの表題などに LaTeX のコマンドが使えるということです. 数式などを表題に含めたい時に便利です. 欠点は, 縦軸の見出しを９０度回転させて表示できないことでしょう.

GNUPLOT の中で,

```
set terminal eepic
set output 'junk.tex'
```

と入力して, これ以降の plot 命令が, LaTeX のソース形式になるようにして, かつ junk.tex というファイルに書き込むようにします. プリンタに出力するときのように, 後は, plot コマンドを入力してください. 一つのファイルに付き一つの plot コマンドにした方が良いでしょう. 別のファイルに切り替えるときは,

```
set output 'junk2.tex'
```

のように新しいファイルを指定します. 全部終ったら,

```
set output
set terminal x11
```

として, 先のファイルをクローズして, 再び画面にグラフを表示するようにしておきます. これで GNUPLOT での作業は終りです.

これを test.tex に取り込むときの書き方を説明します. まず, test.tex の冒頭で

```
\documentstyle[a4j,epic,eepic]{jarticle}
```

のように, epic,eepic スタイルファイルをこの順番で取り込みます. そして文章中の取り込みたい位置で,

```
\begin{center}
\input{junk.tex}
\end{center}
```

というようにして取り込みます. 後は, 通常の LaTeX のファイルと同様に処理してください. 私は, たいていいこちらを使っています. では, 試しに何か書いてみましょう.

6.4.4　プリンタに出力する

画面上で意図したとおりのグラフが表示されたら, これをプリンタから出力してみましょう. これには 2 段階の作業が必要です.

1. GNUPLOT にプリンタが理解できる形式のファイルを作らせる.
2. シェルプロンプトからそのファイルをプリンタに出力する命令を出す.

という手順で出力します. それでは, やってみましょう.

```
set terminal postscript
set output 'junk.ps'
```

このように入力すると, これ以降の plot 命令は, PostScript という規約に従った形式で, junk.ps というファイルに書き込むようになります. このファイル名は何でも構いませんが, PostScript 形式のファイルであることを表すために慣習として, 末尾に.ps を付けます. 後は, これまで通りに plot コマンドを入力してください. 1 つの plot コマンドにつき紙が 1 枚使われます. このとき画面にはグラフはでません. すべてプロットし終ったら,

```
set output
set terminal x11
```

として, 先のファイルをクローズして, 再び画面にグラフを表示するようにしておきます.

では, 次の段階です. まずシェルに戻らなければなりません. 方法はいくつかありますが, ここでは GNUPLOT を終了させることにします. GNUPLOT のプロンプト (gnuplot>) に対して,

```
exit
```

と入力してください. GNUPLOT が終了して, シェルプロンプトに戻ったことと思います. そこで,

```
lpq
```

として誰もプリンタを使ってないことを確認して, [7]

```
lpr junk.ps
```

とします. junk.ps はさっき GNUPLOT で指定したファイル名です. しばらく待っているとプリンタからグラフがでてきます. 結構時間がかかるときもあるので遅いからといって, すぐに lpr をやり直すのではなく, lpq で確認してみてください.

重要 ：もしちゃんと印刷できなかった場合には, 印刷命令を必ず取り消しておきましょう. 印刷命令を取り消せるのは印刷命令をだした本人だけなので, 取り消しておかないと他の人達に多大な迷惑をかけるときがあります. また, 紙資源の節約のためにも, 印刷命令をだした後にいらないことが判明した場合なども積極的に取り消すようにしましょう.

印刷命令の取消方を説明します. まず, 先の lpq を実行してみると, 印刷を待っているジョブ一覧が表示されるので, ユーザーの欄をみて自分の指示した印刷命令を探します. 次にジョブ番号の欄をみて, そのジョブ番号を確認します. 例えば 594 だったとします. ここで,

[7] 本当は誰かが使っていてもいいんですが, ここではこういうことにしておきます.

でみられます. BASIC のユーザーが間違えやすいものとして, べき乗の演算子を挙げておきましょう. べき乗は**です. ^ を使うと, 2 進数としての排他的論理和 (EXOR / XOR) になってしまいます.

なお, 定義域を変更するには,

```
set xrange[0:20]
```

などとし, 対数軸の使用を制御するには,

```
set logscale x
set logscale xy
set logscale y
set nologscale
```

のいずれかを使います. これらの set で行った設定は新たに設定しなおすまで有効です.

6.4.2 離散データをプロットする

次に離散的なデータを折れ線グラフで表示してみましょう. まず表示するデータが必要になります. エディターを使って手入力するなり, プログラムを作って計算させるなりして, 次のように各行に x 座標, y 座標の順に空白で区切ったデータを持つファイルを作ります.

```
10 10
15 30
20 35
30 40
45 60
50 80
```

このファイルの名前が例えば foo だったとします. このデータをプロットするには,

```
plot 'foo'
```

とします. さらに

```
plot 'foo', 0.8 * x
```

のように式と重ねることもできます.

6.4.3 タイトルを付けてみる

グラフにタイトルを付けるには,

```
set title 'Reflection Angle -- Beam Intensitiy'
```

というようにしてから, plot を実行します. これも set コマンドですから設定しなおすまで, ずっと有効です. 消すためには,

```
set notitle
```

とします. 座標軸の見出しは,

```
set xlabel 'input voltage [V]'
set ylabel 'output current [mA]'
set noxlabel
```

などとして制御できます.

6.3.3　拡大

グラフが表示されているところでマウスのカーソルをドラッグすると矩形が表示される．ドラッグを終えるとその部分の拡大されたグラフが別のウインドウになって表示される．

6.4　gnuplot

xgraph は非常に手軽に使えて便利ですが，色々と凝ったことをする場合には，GNUPLOT を使ってみるのもいいでしょう．

次のような時には，GNUPLOT がお勧めです．

- 離散データをプロットしたい．折れ線でつなぎたい．
- 関数をプロットしてその振舞いを確認したい．
- 実験データと理論曲線とを重ねてグラフにしたい．
- $\mathbf{R}^2 \to \mathbf{R}$ の写像を三次元的に表示してみたい．
- 作ったグラフは LaTeX[5] で書いている文書の中に取り込みたい．

6.4.1　関数のグラフを描く

では，X-Window の画面に表示するところから始めてみようと思います．シェルプロンプト[6] がでているところで，

```
gnuplot（特に断らない限り，毎回改行キー押してください．　）
```

と入力すると，オープニングメッセージが表示されて GNUPLOT のプロンプト (`gnuplot>`) がでます．取り敢えず正弦関数でも表示させてみましょう．二次元グラフを描くためのコマンドは `plot` です．

```
plot sin(x)
```

と入力すると，ちょっと時間をおいてからもう一つ窓が開いて，そこに $\sin x$ が $[-10, 10]$ の区間について表示されます．つまり，

```
plot 式
```

とすると，そのグラフが描けるわけです．さらに

```
plot 式1，式2，...
```

として，複数のプロットを重ねることもできます．これらの式で使用可能な組込み関数は，プロンプトから，

```
help functions
```

と入力するとみられます．使用可能な演算子は，

```
help operators
```

[5] LaTeX の基本的な説明は他に譲ります．
[6] 標準環境では `geoph:/home2/s22516>` などと表示されているはずです．

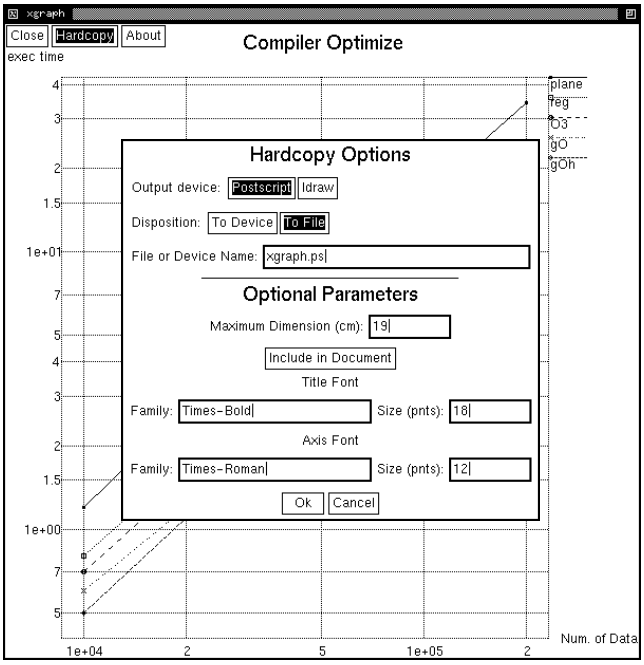


図 6.1: プリントオプション

6.3.2 印刷

左上にある Hardcopy というボタンをクリックするとハードコピーのオプションのウインドウが開く.

Output device

Output device は Postscript と Idraw を選べる.
Idraw を選択すると *idraw* で編集できるファイル形式でファイルを書き出すことができる.
Postscript は postscript 形式のファイルを出力することを指定する.

Disposition

To Device を選択すると直接何かのコマンドにデータを与える.
To File ではファイルにデータを書き出す.

File or Device Name

Disposition が To File の場合, ファイル名を指定する.
To Device の場合, 起動するプログラム名を指定する.

Include in Document

その下は見ての通りの設定である. 重要なものは Include in Document である.
ここを反転状態にしてファイルに書き出すと, TEX の epsf スタイルファイルで扱えるファイルになる.

```
xgraph foo.dat
```

と入力する. すると, xgraph という題名と foo.dat というグラフの名前を持つ折れ線グラフの図を持った X のウィンドウが描かれるだろう.

これで分かるように, xgraph に入力するファイルの各行は X Y の組からなる. もし, 複数のグラフを書きたいのであれば, 各グラフを表示するものの間に空白行を挿入する. さらに, グラフに名前を付けたいのであれば, そのグラフの前に複引用符で囲ったグラフ名を書けば良い.

例として次の入力データについての処理を提示する.

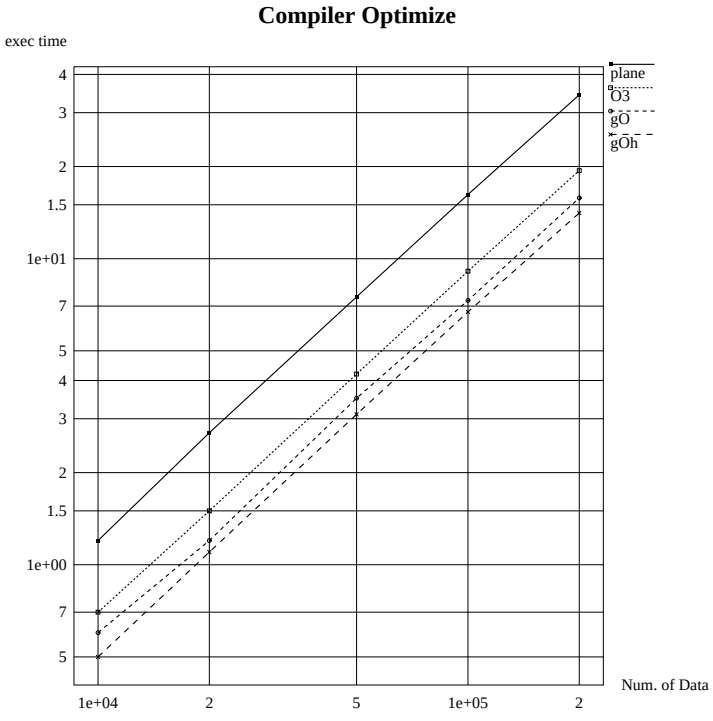
```
TitleText: Compiler Optimize
YUnitText: exec time
XUnitText: Num. of Data
LogX: true
LogY: true
Markers: true
BoundBox: true

"plane"
10000 1.2
20000 2.7
50000 7.5
100000 16.2
200000 34.3

"O3"
10000 0.7
20000 1.5
50000 4.2
100000 9.1
200000 19.4

"gO"
10000 0.6
20000 1.2
50000 3.5
100000 7.3
200000 15.8

"gOh"
10000 0.5
20000 1.1
50000 3.1
100000 6.7
200000 14.1
```



では，実際にやってみましょう．{\em idraw} で {\tt pic/texeps.eps} を作りました．上の通りにここに図を入れます．

```
\vspace{1cm}
\begin{center}
\epsfile{file=pic/texeps.eps}
\end{center}
\vspace{1cm}
```

6.2 draw tools

6.2.1 idraw

idraw は Macintosh 上にある *MacDraw* ライクな drawing tool である．*tgif+*との大きな違いは漢字，アルファベットを問わず文字に対する拡大縮小，任意の角度の回転が可能なことである．

反面，*idraw* は線の太さと形状，フォントの大きさと形状，の概念が分離していない，*kinput2* を自動的に立ち上げられないという弱点を持つ．

idraw を立ち上げる前，または立ち上げた後に kinput2 &とバックグランドで *kinput2* を立ち上げれば漢字を入力することができる．

6.2.2 tgif+

*tgif+*はどちらかというと *MacDraw2* に似た感触の線画ツールである．*idraw* に比較したときの長所は *kinput2* の自動起動がある，xbm 形式のビットマップデータを取り込める，線や字体の概念の直交性が良い，懇切丁寧なマニュアルが存在すると言うことである．

6.3 xgraph

xgraph は 2 次元グラフを描くのに適している．しかし，グラフといっても数式によって与えられたグラフを描くことはできない．可能なのは折れ線グラフと棒グラフである．

特徴は *idraw* に画像としてではなく，線画データとして取り込めるということである．これによって様々な装飾などが可能となる．勿論，eps 形式でも出力できるので TEX にも取りこめる．この時は Hardcopy のところで include in Document を指定する．

6.3.1 折れ線グラフ

まずは簡単な折れ線グラフを描くことを示す．

foo.dat というファイルを emacs などのエディタで作る．foo.dat の中身は次の通り．

```
10000 1.2
20000 2.7
50000 7.5
100000 16.2
200000 34.3
```

さて，これで，シェルの命令待ちの状態で

emacs の中から書いてあるとおりに操作しているのにうまくいかない 地物標準環境がうまくインストールされていない可能性があります. コマンドラインからの操作手順でやってみるといいかもしれません.

LaTeX がキャパシティーを越えたといって終了した ここにインストールしてあるのは, 既に Big TeX ですから, これでだめならどうしようもありません. 分割するなどの工夫をしてください.

いくら待ってもプリンタから出力されない 出力は以外と時間がかかります. 特に一枚目は, 5 分ぐらいかかるときもあります. `lpq` 命令で確認してみてください.

6.1.7　LaTeX 文章に絵を取り込む方法

レポートなどでは図やグラフを文章の中に入れることが必要となります. ここでは LaTeX 文章の中に図を入れることを示します.

まず, 図ですが, 図は *idraw* や *tgif+* で書いていて下さい. *idraw* ではそのまま, *tgif+* では *eps* ファイルを書き出すように指定すれば LaTeX 文章に取り込むことができます.

では, 次に LaTeX 文章の方の準備を説明します. `jarticle` で書くとすると文章の`\documentstyle` を次のように指定します.

```
\documentstyle[epsf]{jarticle}
```

もちろん, `jreport` で書く時には `jreport` と指定していて下さい. 次に, 図を入れたい場所に

```
\epsfile{file=FILENAME}
```

と書きます.

では, 実際にやってみましょう. *idraw* で `pic/texeps.eps` を作りました. 上の通りにここに図を入れます.

xdvi でプレビューすると図のところに四角の枠が出てきます. `C-g` で図が出てきます. もう一度 `C-g` を押すと図が消えます. すなわち, `C-g` は図を出すかどうかを決めるモードの切替スイッチになっています. プレビューの時に図を出しておくと時間がかかるので, 普段は図を出さないモードにしておくと良いでしょう.

この `dvi2ps` 出力ならば, 図入りの文章でも適当な方法で E-mail に入れて誰にでも送りつけることができます.

以下は, 上の文章で図を採り入れている部分の LaTeX 原稿です.

```
dvi2ps test.dvi | lpr -h
```

のようにパイプでつなぎます.

重要 ：もしちゃんと印刷できなかった場合には, 印刷命令を必ず取り消しておきましょう. 印刷命令を取り消せるのは印刷命令をだした本人だけなので, 取り消しておかないと他の人達に多大な迷惑をかけるときがあります. また, 紙資源の節約のためにも, 印刷命令をだした後にいらないことが判明した場合なども積極的に取り消すようにしましょう.

印刷命令の取消方を説明します. まず, コマンドライン[4] から

```
lpq
```

と実行します. すると, 印刷を待っているジョブの一覧が表示されるので, ユーザーの欄をみて自分の指示した印刷命令を探します. 次にジョブ番号の欄をみて, そのジョブ番号を確認します. 例えば 594 だったとします. ここで,

```
lprm 594
```

というようにすると印刷命令が取り消されるはずです. もう一度 lpq を使って, 本当に取り消されたかどうか確認してください. 自分が指定した印刷を全て取り止めるときは,

```
lprm s22516
```

のように自分の ID を指定します.

6.1.5 MS–DOS で作ったソースをコンパイルするときは

MS–DOS で作ったソースファイルを Unix に持ってくる方法はここでは説明しません. ftp というキーワードだけ挙げておきますので他のガイドを参照してください.

MS–DOS のファイルは, 通常シフト JIS コードで書かれているので, これを EUC コードに変更する必要があります. 変換方法は, 一つは emacs で読み込んで少し書き換えてから (例えばスペースを一つ挿入してからすぐ削除) セーブし直す方法です. もう一つは,

```
mv test.tex test.tex.old
nkf -e test.tex.old > test.tex
rm test.tex.old
```

とする方法です. nkf がない場合は kconv を試してみるといいでしょう.

6.1.6 うまくいかないときのヒントをいくつか

あってるはずなのにエラーがでる こういう時はたいていなにか勘違いをしているだけのはずですが, ファイルの先頭付近でででるんでしたら, 漢字コードは合っていますか ?とだけ書いておきましょう. 漢字コードは EUC でなければなりません. 特に MS–DOS から転送してきたファイルの場合に忘れがちでしょう.

LATEX の解説書通りのはずなのにエラーがでる 解説書は ASCII 版の解説書ですか ?NTT 版の解説書だった場合, 一部のファイル名とコマンドが異なります. ポピュラーなものとしては, ASCII 版では, j-article ではなく jarticle を使います.

[4] 地物標準環境では geoph:/home2/s22516> などと表示されているはずです.

6.1.2　LaTeX 処理をしてみる.

書きあがったら LaTeX を通してみましょう. 地物標準環境ではソース (test.tex) の文章中にカーソルがある状態で, C-c C-a と入力すると, 画面が二つのウインドウに分かれて, 一方には test.tex が表示され, もう一方には LaTeX からのメッセージが表示されます.

エラーがあった時は, ミニバッファに

```
Formatting:  ERRORS
```

という表示が出ます. C-c C-n とすれば, エラーメッセージとその説明が画面の半分に出され, もう半分にはエラーのあった箇所が表示されます.

シェルのコマンドラインからコンパイルしたい場合は, コマンドは,

```
jlatex test.tex
```

もしくは単に

```
jlatex test
```

です. これで, test.aux, test.log, test.dvi ができます.

6.1.3　xdvi を使ってのプレビュー

エラーなく LaTeX 処理が終ったら意図したとおりに組版されているかどうかを確認することになります. 地物標準環境では, ソースの文章中にカーソルのある状態で C-c C-p とすると xdvi というプレビューアーのウインドウがしばらくして開きます. そこに今処理した文書が表示されます.

見ただけでも使い方は分かると思いますが, ウインドウの右の方にあるボックスをマウスで左クリックすると, それぞれの機能が実行されます. 一番良く使うのは Next(次頁) と Prev(前頁) でしょう. Page+5, Page+10, Page-5, Page-10 を使うと 5(10) 頁後 (前) に行けます. Shrink1, Shrink2, Shrink3, Shrink4 で大きさが変わります. Quit でプレビューが終ります.

なお, シェルのコマンドラインからプレビューアー xdvi を立ち上げたい時は, コマンドは,

```
xdvi test.dvi
```

もしくは単に

```
xdvi test
```

です.

6.1.4　プリンタへの出力

最後にプリンタへ出力します.

プリントアウトするためには, まず, ps (PostScript) ファイルに変換しなければなりません.

```
dvi2ps test.dvi > test.ps
```

これを印刷するには,

```
lpr -h test.ps
```

とします. ps ファイルが必要なければ,

というものです. 入力漢字コードは EUC です. うまくいかなくなって人に質問する場合に必要になるかもしれないので気にとめておいてください.

それでは, ソースを書くことからプリンタに打ち出すまでひと通り説明してみます. 以下は地物標準環境での説明で, X–Window はすでに起動しているものとしています.

6.1.1 ソースを書く

ここのワークステーションで漢字を入力するためには, emacs[1] を使います. emacs をたちあげてください. .tex で終るファイル名を決めて (例えば test.tex) C-x C-f[2] と入力すると, 最下行にファイル名を入力するようになるので, test.tex と応えてください.

地物標準環境では, .tex で終るファイル名を指定すると自動的に漢字コードを EUC にして LATEX のソースの入力に適したモードになるように設定してあるはずです. 具体的には,

```
(setq LaTeX-mode-hook
'(lambda ()
(setq kanji-fileio-code 3)
(setq TeX-command "jlatex")
(define-key LaTeX-mode-map "\t"
'TeX-complete-symbol-or-LaTeX-indent-line)
(define-key LaTeX-mode-map "\e\t" 'LaTeX-indent-line)
(setq LaTeX-item-indent 6)
(boil)
))
```

というようなものが .emacs の中にあるはずです. よって白黒反転した行が

```
-----NEmacs:  test.tex (EJJ-:LaTeX Fill)--All------
```

という感じになっているはずです. 重要なのは LaTeX というのがあることと, その左のアルファベット 3 文字の一番目が E であることです. LaTeX でないときは, M-x latex-mode[3] とします. E となってないときは, C-x C-k f を何回かして E とします.

後は, emacs の説明書と LATEX の説明書をみながらソースファイルを書いてください. 一つだけ便利な機能を紹介しておきます.

```
\begin{document}
```

のように begin のある行にカーソルがあるときに C-c C-f と入力すると,

```
\begin{document}


\end{document}
```

というように対応する end の行を作ってくれます.

書きあがったら C-x C-s でセーブしておきましょう.

[1] エディターの名前です. 詳しい説明は 2.9 節に譲ります.
[2] C-x は control キーを押しながら x を押すと言うことです.
[3] M-x は, メタキー (スペースバーの横のダイヤのマークのキー) を押えたまま x を押すこと, または, ESC キーを押してから x を押すことです. その後最下行で latex-mode と入力して改行キーを押します.

第 6 章

文書の作成と描画

わたくしが文学と言いますのは, 一群の, 一連の作品のことではなく, 商業や教育の一部門のことでもなく, 一つの実践, 書くという実践によって残された痕跡からなる複合的な書き物のことなのです.
Roland Barthes　就任講義」

世の中の計算機の大部分の使われ方は文書処理です. もちろん, UNIX Workstation も例外ではありません. それどころか, UNIX Workstation は非常に高度な文書処理用のプログラムが存在しています.

まずは文書を清書するプログラムの TEX を説明します. 現在, 世界中で数学や計算機科学のレポートや論文の大部分はこの TEX を使って書かれています. このテキストとて例外ではありません.

次に 2 種類の draw tool を紹介します. 文書や OHP を作る際に大変便利です.

その次に 2 種類の graph tool を説明します. 数値実験のプログラムがこれらの graph tool が解釈するような出力をするようにプログラムされていたならば, 非常に簡単に結果をグラフ化することが可能です. 実験などの結果をグラフにまとめるときにいろいろとコンピュータに計算させることも多くなるでしょう. それらを計算機上で操作し, graph tool にかけることは計算機の特性を考えると非常に自然な作業でしょう.

また, 当然にもこれらの結果は TEX に取り込めます. それぞれに違った性格を持つプログラムであるので用途によって使い分けることが可能でしょう.

現在世界中で, Computer Network を通じて論文や, テクニカルレポートが配布されています. 出来てから数時間しか経っていないそれらを身近なプリンタで出力して読むことができます. このような時にグラフや図が文章の中に電子的に埋め込まれているというのは非常に大切なことです. 糊で後から貼りつける操作を Compter Network を通じてはできないからです.

6.1　TEX& LATEX

TEX 自身については最近は随分書籍がでてますのでそちらに譲ることにしてここでは, TEX の使い方でローカルなことについての説明を行います. 一般的なマニュアルには, 例えば次のようなものがあります. TEX そのものに関しては, 分厚くて読みにくいが製作者自身の筆によるバイブルクヌース, 1989 や一通り概観的に書いてある大野, 1989 などがある. LATEX については, 製作者自身によるマニュアルランポート, 1990 や日本語版も含めた入門書奥村, 1991 などがある.

まずここにインストールしてある TEX のバージョンです. バージョンは ASCII 版の

```
BigTeX, C Version 2.99 - j1.7e (no format preloaded)
LaTeX Version 2.09 <24 May 1989>
```

```
    2 dict begin  % 2 個のローカルな辞書を用意する .
    /angle exch def  % angle =  (スタックの 2 番目のもの ) と定義
%このときスタック 2 番目にあったものは消える .
    /length exch def % length
    length angle cos mul  % x 座標 length * cos(angle)
    length angle sin mul % y 座標 length * sin(angle)
        rlineto % (x,y) だけ移動する
    end % ローカル辞書の終り
} def
/c-curve {
    2 dict begin
    /angle exch def
    /length exch def
    length min-length lt
% 条件判定 length < min-length なら true を返しておく
        {length angle plot-line}
      % あまりに length が小さい時のためのルーチン
        {length 2.0 sqrt div angle 45 add
             c-curve % 自分を呼んでいる
         length 2.0 sqrt div angle 45 sub
             c-curve}
    ifelse % lt が true なら{最初} false なら{ 2 番目}
    end
} def
/min-length 10 def
newpath
400 250 moveto 250 90 c-curve %c curve を書く
stroke
showpage
```

Network News (4.3 節参照) の comp.lang.postscript にときどき面白い PostScript のプログラムが投稿されます . 何か面白いプログラムができたら投稿してみると良いかも知れません.

また , 日本語も扱うことができます . 例えば, EUC コードのエディタを使って書いている場合は, フォント名を/Ryumin-Light-EUC-H などとしておけば良い.

```
%(3) self reference
/Times-Roman findfont 18 scalefont setfont      %文字のサイズ (後で縮小)
/draw {20 20 moveto 560 20 lineto 560 820 lineto        %サブルーチン draw
20 820 lineto 20 20 lineto              %枠を書き
25 800 moveto                           %moveto で指定した位置に () 内を記録する
(   One day a student came to Moon and said, "I understand how to make) show
25 775 moveto                           %() の文字列をスタックに置き show
(a better garbage collector. We must keep a reference count of the pointers)
show
25 750 moveto
(to each cons." Moon patiently told the student the following story:) show
25 725 moveto
(   ") show
25 700 moveto
(   One day a student came to Moon and said, "I understand how to make) show
25 675 moveto
(a better garbage collector. We must keep a reference count of the pointers ...) show
} def                                   % /name { } def で定義
newpath
draw                                    %  draw を呼ぶ
stroke
newpath
54 80 translate                         %以下の絵の原点を 54,80 に移動
0.8 0.8 scale                           %全対を 8 割に縮小
draw
stroke
newpath
54 80 translate                         %縮小,  移動を繰りかえす
0.8 0.8 scale
draw
stroke
newpath
54 80 translate                         %縮小,  移動を繰りかえす
0.8 0.8 scale
draw
stroke
showpage                                %4 回書いたら出力する
```

(3) は Lieberman , Hewitt の garbage collector の論文にある引用です . 有名な AI の公案のひとつです . Moon は Lisp Machine を作った MIT の David Moon のことです .

再帰呼び出しも可能ですので , (4) のような C curve を出力することもできます .

```
%!
%(4) c-curve, a recursively defined curve
/plot-line {
```

のようになっています．美しいでしょう．

このような絵の座標をきめるのに座標系の書かれた方眼紙があると便利です．これは，(2) で得られます．

```
%!
%(2) grid
/Times-Roman findfont 10 scalefont setfont      %文字を準備
/nstr 3 string def
/prt-n {nstr cvs show} def      %サブルーチン prt-n はスタック上の整数を
newpath                         %current position に記録する
[3] 0 setdash                   %破線を書く準備
1 setlinewidth                  %太さを 1 にセット
0 20 595 {0 moveto 0 835 rlineto} for   %制御変数を 0;20;595 で{}内を実行（横線）
0 20 835 {0 exch moveto 595 0 rlineto} for      %同様に縦線を引く
stroke                          %記録する
[] 0 setdash                    %実線を書く準備
2 setlinewidth                  %太さを 2 にセット
0 100 595 {0 moveto 0 835 rlineto} for          %100 毎の横線
0 100 835 {0 exch moveto 595 0 rlineto} for     %100 毎の縦線
stroke                                          %記録する
0 100 835 {dup 22 exch 3 add moveto prt-n} for  %100 毎の縦座標を書く（注参照）
600 0 translate                                 %回転する（注参照）
90 rotate
0 100 595 {dup 22 exch 5 add moveto 600 exch sub prt-n} for     %横座標
840 0 translate                                 %さらに回転する
90 rotate
0 100 835 {dup 22 exch 3 add 40 add moveto 800 exch sub prt-n} for
600 0 translate                                 %上は右端の縦座標を書く
90 rotate                                       %回転する
0 100 595 {dup 22 exch 3 add moveto prt-n} for  %上端の横座標を書く
showpage                        %出力する
```

（注） 0 100 835 {dup 22 exch 3 add moveto prt-n} for は制御変数として 0,100,200,...,800 をスタックに置いては内を実行する． 100 の場合はまず dup でスタックトップを 100,100 にする．次に 22 をのせ exch するからスタックは 100,22,100 になる． 3 add でスタックトップは 22,103 になりそこへ moveto する． スタックには 100 が残っているがこれが prt-n で記録される． つまり x=22，y=103 の位置に 100 と書く．同様に 200,300,... を書く．

```
600 0 translate
90 rotate
```

は 90 rotate でこれから書く絵を原点を中心に 90 度反時計まわりに回転し（第 2 象限に移る）それを 600 0 translate で x 正の方向に 600 移動する（第 1 象限に引き戻す）指令．これで下端に座標を書く．以下同様．

回転，移動だけでなく，拡大，縮小もできます．(3) をみましょう．

```
%!
```

x x x という順番のスタックに対して，後ろ二つを mul して x^2 +3 を作り出します．そして x x^2 +3 というスタックをイクスチェンジすると，x そのものがスタックトップに来るわけです．

細かい手続きを登録しておく場所が辞書スタックです．手続きを登録しておきたいときなどは

```
\procedure{
    うんたら
} def
```

のように書いてやると，つぎにくる {から }def までの間に書かれた手続きがすべて辞書の中に手続きとして登録され，知らない関数名などが現れるとシステムはスタックの上から順番に探していきます．これを使いたいときには簡単に

```
procedure
```

と書くことによって呼び出されます．

たとえば，PostScript の座標系ではデフォルトが 1 インチ = 6 4 ポイントに当たりますが，これをいちいち計算するのがめんどくさいとき，

```
\inch{ 64 mul }def
```

と書いておけば，現在点から右へ 2 インチの直線を書く，といった手続きは

```
2 inch 0 rlineto
```

と表されます．こうやっておくと，PostScript 内部で

```
2 64 mul 0 rlineto
```

と変換されてしまうわけです．

PostScript の画面は標準では座標にして x=0 から x=600，y=0 から y=840 の範囲が縦長の A4 の紙に出力されます．　左下が原点．

詳しいことは，アスキー出版から出ている「PostScirpt チュートリアル & クックブック」アドビ・システムズ, 1989 を参照してください．これは 3 冊セットになっている PostScript マニュアルのなかの青色の本です．

5.8.3　サンプルプログラム

以下に, サンプルを挙げます.

PostScript のプログラムは第 1 行に %! を書きます．%はコメント開始記号です．(1) は

```
%!                                  PostScript のプログラム
%(1) y = x^2
newpath                             %軌跡を書く用意をする
60 240 moveto 540 240 lineto        %60,240 から 540,240 まで X 軸を書く
300 720 moveto 300 120 lineto       %300,720 から 300,120 まで Y 軸を書く
60 720 moveto                       %60,720 に curent position を移動
220 80 380 80 540 720 curveto       %220,80;380,80 を参照し 540,720 まで 2 次曲線を書く
stroke                              %今まで書いた軌跡を (太さ 1 で) 記録する
showpage                            %紙に出力
```

5.8 PostScript

各端末機室にあるページプリンターはテキストファイルだけではなくいろいろな図形をプログラムしてプリントアウトすることができます．そのためのページプリンタ専用の記述言語が Adobe という会社から出ています．それが，PostScript です．最近の業務用のプリンタで PostScript 対応と書かれているものはすべてハードディスクを内蔵していてこの PostScript を積んでいます．(だからむやみに電源を切ってはいけません．クラッシュの原因となります) これを用いることによって，直線や曲線，いろいろなフォントをクリップ (型どり) してその中に絵を書く，といったことができます．TEX などで書かれたきれいな原稿を手にしたことがあるでしょうか．dvi2ps で出力されたファイルは，例えば，foo.ps という名前になっていると思います．これで PostScript のコマンドに直されて出力されているのがわかります．テキストが手元にあればのぞいてみて下さい．

5.8.1 PostScript の書き方

PostScript などという実行型プログラムは存在しません．%! で始まるファイルを lpr コマンドでプリントアウトするとプリンタが PostScript の命令を解釈してイメージしている図形を出してくれます．どんな命令を使っているか知りたいとき，つまりベタのプログラムを知りたければ画面上に cat, more,less で表示します．プリントアウトはファイルの先頭の%! を消してしまって lpr コマンドをすれば，ファイルがそのままの形でてきます．

5.8.2 特徴

プリンタはそんなに賢くありませんからデータを蓄えるのは苦手です．データはすべてスタックに入れられてしまいます．つまりグローバル変数はありません．

例えば足し算でも

```
1+1=2
```

ではなく PostScript に対して 1 1 + といれてやって

```
2
```

という一つのデータに置き換えるという操作になります．+ (ほんとうは add) というオペレータがその直前に入れられた 1 1 という二つのデータをポップしてきて改めて，その結果の 2 をスタックにプッシュするという動作になるわけです．

では

```
(x^2+3)*(x+2)
```

はどうやって表せばいいでしょうか．これは

```
x dup mul 3 add x 2 add mul
```

ここで dup はスタックの一番上の x をデュプリケートしてもう一つ x というデータをスタックに積みます．この x と x をかけ算してできた x^2 というデータに対して 3 を積んで add ，その上に x+2 を表すデータをつくって，mul しています．実際には，x というデータは 1 回しか与えられないことが多いですから予めもう 1 回 dup しておいてスタックの順番を入れ換える，といったことが行われます．

```
x dup dup mul 3 add exch 2 add mul
```

- サブルーチン群の “スイッチ”. 例えば, common 変数 switch の値を 'DISPLAY' にすると一連の出力サブルーチンは結果を画面に書き出すが, 値を 'FILE' にすると結果をファイルに書き出す, などのように, 複数のサブルーチンの動作を統一的に制御する一種のスイッチとして common 変数を使うことができる.

この, 後の 2 つの使い方[29]では, common 変数に値を代入する場所を一部に限り, 他の部分では定数として扱うことが大切である.

非標準の機能は注意して使うこと

多くの処理系は, 標準の言語仕様に沿った機能の他に非標準の機能を用意している. 例えば, 先に述べた implicit NONE は FORTRAN 77 という規格においては標準ではない. 標準の機能だけを用いたプログラムなら, (原則的には) パソコンで動いているものを大型計算機に持っていってもそのまま動く. しかし, 非標準の機能を使うとそのような移植性は犠牲になる.

よって, 非標準の機能を使う場合にはそれと意識して使い, 移植のためにその機能が使えない場合にどこを変更すれば良いか分かるようにしておかなくてはならない.

安全なプログラムを書くこと

異常事態が起こった時にも安全側に「倒れる」ようなプログラムを書くべきである (“fail-safe”).

例えばサブルーチンであれば, 不当な入力があったときにはエラーメッセージを出してその場で停止するなどの安全な設計が望ましい. 自分が作って自分で使うサブルーチンでも, そのサブルーチンを呼ぶ以前の段階のエラーでそのサブルーチンの引数に不当な値を渡してしまうことはよくある. そのような時, そのサブルーチンが不当な値の時にでたらめな動作をしてしまうなら, エラーがそのサブルーチンにあるのかそれ以前にあるのか判りにくくなってしまう. もし, 不当な値の時にエラーメッセージを出して止まるようになっていれば, エラーはそのサブルーチンではなくそれ以前にあることが判る. 不当な値が入ってくるかも知れないと思ってサブルーチンを書くことが大切なのである.

このように, 異常事態をどんどん検出しはじき出すようにしておけば, debug の時役立つし, 後にプログラムを変更する時にもおかしな動作に悩むことは少なくなるであろう.

その他

総じて, 言語仕様の細かい点に関わるような書き方, 危なっかしい書き方は避けるべきである. ここまでにも何回か同様なことを述べた.

演算子の優先順序を覚えておくよりも, 括弧を用いて陽に演算順序を指示すべきである. (例えば, f**g**h の演算順序などという言語仕様の隅っこをつつくのは不毛である.) ただし, 四則演算の優先順序は余りにも当たり前なので, これは無理に括弧を使う必要はない.

Do 文の制御変数が do-loop の後どんな値をとっているか, などはやはり言語仕様の細かい点であるので, それに依存するようなプログラムを書くべきではない.

解り易さや安全性を犠牲にした効率化は, よっぽど必要でなおかつ効果が見込める時にのみ行なうべきである. 例えば, 実数型配列の全ての要素を 0.0 に初期化したいとしよう. FORTRAN 77 の規格では初期化していない変数の値は実行開始時には不定となることになっているが, 実行開始時に 0 に初期化してくれる処理系もある. そのような処理系ではわざわざ 0.0 に初期化するのは非効率的に思えるかも知れないが, その初期化に費やす時間は全体のごく僅かな部分であろうから, 他の処理系では正しく動作しないかもしれない危険を犯すほどの価値があるとは思えない.

[29] これらは, entry を使って一つの subroutine をサブルーチン群のように扱うことで解決する場合も多い.

実行文中に “マジックナンバー” を埋め込まないこと

定数を実行文の中に埋め込むのは良くない. FORTRAN なら parameter 文で, C なら define で名前をつけてその名前で参照するべきである. なぜなら, 実行文中に

```
if  (f .gt. 1.e7)  then
```

のように現れた定数は, 他の人にとっては “マジックナンバー”(何を意味するのか分からない数) である. もし

```
real        DATMAX
parameter  (DATMAX = 1.e7)  /* maximum value of the data */
   .........
if  (f .gt. DATMAX)  then
```

のように書かれていれば, この 1.e7 という定数の意味ははるかに分かり易くなる.

また, その定数を変更する時には, 名前をつけておくやり方の場合は一箇所を変更するだけで良いが, 実行文中に埋め込まれている場合, その定数を一々探し出して変更しなくてはならないので間違いが起こり易い.

さらに, そのような名前の宣言はプログラムのなるべく小数の場所に集中させるのが良い. これも, 変更する部分が一部に集中していた方が安全に変更できるからである.

総じて, プログラムは一部を変更するだけで安全に目的の変更が達せられるように設計するべきなのである.

サブルーチンを解り易さのために使うこと

何度も同じような処理をする場合にその部分をサブルーチン化しようというのは誰でも思いつくことであるが, 1 回しか呼ばれないサブルーチンもプログラムの構造を見易くするために役立つことが多い. プログラムのまとまり毎にサブルーチン化しておけばそのまとまりの独立性が高くなり, 後に変更したくなった時にもどの部分を変更すれば良いか解り易く, また, 変更の副作用が他の部分に及ぶ可能性も減る.

FORTRAN の common 変数はなるべく使わないこと

Common 変数は無闇に使わないようにするべきである. 特に, サブルーチンの引数の代わりにするようなことを絶対にしてはいけない. Common 変数は見えない所で別の値が代入されているかも知れないので危険なのである.

Common 変数は次のような使い方に制限すべきであろう:

- サブルーチンの work area. いろいろなサブルーチンが work area を必要としているが全てをそのサブルーチンの local 変数にするとメモリが足りない, というような場合, work area を共有するために common を使うと良い[27]. その場合にも, その common の領域をサブルーチン間の値のやりとりに使ってはならない.
- プログラム単位にまたがる定数でしかもコンパイル時には決まらないもの[28]. プログラムの最初の部分で外部から読んだ値をそれ以降定数として使いたいというような時, その値を common 変数に格納しておくと良い場合がある.

[27] もう一つの手は引数として work area を渡してやることだが, 余りスマートとはいえない.
[28] コンパイル時に決まるものは別のファイルに parameter 文で定義しておき, そのファイルを各サブルーチンで include するとよい. ただし, include 文は非標準であるので注意.

Goto 文はできるだけ使わないこと

Goto 文は, 多くの場合, プログラムを解りにくくする元となる.

FORTRAN 77 には, C や Pascal では当たり前の while 文などが用意されていない. そこで, while-loop などのよくあるループを構成するのに goto 文が必要となる. しかし, goto 文を使うのは 「よくあるループ」に限るべきである. そして, 上の 例 2 のように, どのようなループかをコメントによって示すと良い.

C には 「よくあるループ」は既に用意されているので, ループのための goto 文は必要ではない. ループの途中で抜けるためには break 文が使える.

その他, エラー処理など goto 文を使った方が分かり易い例もあるが[25], それは例外であり, ほとんどの場合は goto 文は使わない方が良い.

変数の型を常に意識すること

変数の型を強く意識した書き方を常にするべきである.

プログラム中の式はコンパイラーに対する指示であって, 数学の式ではない. 例えば, 1.0 と 1 はプログラム中では異なった意味を持つ. 従って,

```
i = j / 6      /*  i, j は integer  */
f = g / 6.e0   /*  f, g は real     */
```

のように, 定数の型は明確に表現するのが良い.

さらに, 型変換はコンパイラーに任せないで, 型変換関数を用いて陽に指示するようにする方が良い. 例えば,

```
f = i / j * 3.5e0    /*  i, j は integer  */   /* (a) */
```

と書くと, ちょっと見ただけでは

```
f = real(i) / real(j) * 3.5e0                  /* (b) */
```

と

```
f = real(i / j) * 3.5e0                        /* (c) */
```

のどちらなのか分かりにくい. (a) は, 実は (c) のように計算されるが, 不注意なプログラマーが (b) のつもりで書いたのかも知れない. (例えば `i = 2, j = 3` の時, (b) と (a)(c) では `f` の値は違うことに注意.) (a) と書けば (c) のように演算されるという型変換の規則を覚えておくよりは, 陽に (c) のように書く方が誤解の余地が少なくなり望ましい.

FORTRAN でも変数は必ず宣言すること

FORTRAN には, 名前が i – n で始まる変数は宣言しなくても integer となる, などの暗黙の型に関する規則があるが, 全ての変数は宣言してから使うことを勧める. そして, 宣言していない変数を使おうとするとコンパイラーが警告してくれるように, 最初に `implicit NONE` を宣言しておくと良い[26].

変数を宣言しないで使っていると, 変数名の綴を間違えた時もコンパイラーは警告をしてくれないので, 思わぬトラブルの元になり得る.

ただし, 宣言をしないスタイルになれている人との間での誤解を避けるために, 暗黙の型にしたがった名前を使いながらなお宣言するのが, 我々が現在おかれている環境では一番良いやり方だと思う.

[25] このような例については カーニハン, リッチー, 1981 を参照のこと.
[26] `NONE` は FORTRAN 77 標準ではないが, 東大の大型計算機上の FORTRAN や Sun FORTRAN では許されている.

タイプの時間を惜しまないこと

この項目より後の各項目で述べる注意事項には, それに従えばタイプする文字数が増えるようなものが多い. しかし, タイプを面倒くさがってはならない.

あるプログラムを書き始めてからそれが望み通りの動作をするようになるまでの時間のうち, 1/3 が実際にプログラムを書いている時間で残りの 2/3 は debug とテストの時間であるという. (大きなプログラムほどこのことは正しいようだ.) とすれば, debug やテストを効率的にできるようなプログラムを書くことが最終的には得なのである. さらに, 実際にプログラムを書いている時間のうちタイプしている正味の時間は非常に少なく, 手を止めて考えている時間の方が多いであろう.

従って, タイプの手間を惜しまず, 効率的に debug やテストができるように, 解り易いプログラムを書くことを心がけるべきである.

コメントをたくさん書くこと

その場限りの使い捨てのようなプログラムでない限り, source program 中には他人が読んでも解るように十分なコメントを付けるべきである.

例えば, 変数の宣言の後にその変数の意味を書いたり, また, 一連の手続きの前にこれから何をするのかを簡単にメモしたりしておけばプログラムを理解するのに非常に役に立つ.

また, 空白行を適度に入れてプログラムの構造を見易くするのも良い習慣である.

字下げをすること

プログラムの論理的構造にしたがって字下げをする習慣を, 是非身につけるべきである. (これは, 既にほとんど常識となっていてほとんどの人が身に付けていることではあるが.)

例えば, DO ブロックなどは

```
/* 例 1 */
     do 100  i = 1, 10
        sum = sum + i
        write(*,*)  sum
100  continue
```

のようにブロック内を字下げするべきである. また, 言語仕様で用意されているブロックばかりでなく, プログラムする人が意識する論理的ブロックも, 例えば,

```
/* 例 2 */
100 continue                        /* repeat */
        sum = sum + g
        g = g / 3.0
     if  (g .gt. eps)  go to 100    /* until (g <= eps) */
```

のように字下げすると良い.

の FORTRAN の例.), FORTRAN 77 標準ではないのでここでは仮にこのようにしておく. なお, Sun FORTRAN では `i = 1 !コメント` のように書く.

自分のプログラムを人に見せたり使わせたり, あるいは, 人のサブルーチンを使ったりする場合には, 当然, 周囲の人が使っている言語に合わせる必要がある. プログラムの規模が大きかったり, 同じプログラムを長期に渡って使用したりする場合などに, このような必要が生じやすい.

また, 周囲の人が使っていれば, 数学サブルーチンパッケージのようなソフト資源も手近にあるであろうし, マニュアルも整っているであろう. さらに, 周囲の人に分からないところはすぐに教えてもらえるだろう. このようなことが, 実際コンピューターを使っていく上では意外と重要なのである.

地球惑星物理学科関連者の現状では FORTRAN を主に使う人の方が多いので, FORTRAN を使わずに済ませることは事実上難しいかも知れない.

次に, 計算を支援する環境について述べる.

気象や海洋の大循環モデルのように, 現在のコンピューターの能力ぎりぎりの計算をする場合には, スピードが決定的に重要になる場合がある. そのような場合はスーパーコンピューターを使うことになるが, 現在の東大の計算機センターのシステムでは, FORTRAN でしかスーパーコンピューターを使うことはできない[23]. スーパーコンピューターに対応した C コンパイラーが導入されるのはそんなに遠い将来ではないが, 現在の FORTRAN コンパイラーほど優秀になるにはかなり時間がかかるかも知れない.

行列計算のような数値計算のサブルーチンパッケージや, 諸々のソフトも (歴史が長い分) FORTRAN の方が整っているし, 信頼性も高いと思われる.

最後に, 結論を述べる.

現在我々(少なくとも地球惑星物理学科関係者) がおかれている状況では, FORTRAN を避けようとする人には, FORTRAN を自然に使う人よりも多くの努力が必要となる. 場合によっては避けられない. 従って, FORTRAN でなければできない仕事や FORTRAN の方が楽な仕事 (多分, 研究グループ規模の仕事) には FORTRAN, それ以外 (多分, 小規模, あるいは, 個人で閉じている仕事) には C と使い分けるのが最善であろう. 次善ではあるが, 言語は 1 つだけしか使いたくないというなら FORTRAN を選ぶべきである.

「現在我々がおかれている状況」も少しずつ変わっていき, これからはますます C の勢力が強くなってくるのは間違いない. だから, これからコンピューターと長くつきあっていく可能性のある人は, 積極的に C も使うことをお勧めする.

5.7　良いプログラムの書き方

この節では, 初心者が良いプログラムを書くために注意すべき点を述べる.

良いプログラムの第 1 の条件は,

　他人にも分かり易い (従って, 自分が後で読んでも分かり易い) こと

である. もちろん, 場合によっては分かり易さよりも速さをとらなくてはならないこともあるが, 分かり易さを第 1 の条件にしたほうが最終的には得である場合が多い.

プログラムの全ての面における注意を与えるだけの力は筆者にはないので, ここでは主にプログラムの見かけ (表面的なこと) についての話題に絞る. 深く勉強したい人は カーニハン, プローガー, 1982 を参照して欲しい.

以下に, いくつかの項目について望ましいプログラムの書き方を述べる.

ここでは, 主に FORTRAN を使って例を示す. ただし, FORTRAN であっても C にならって

```
/* このようにくくってある部分はコメント */
```

とする[24].

[23] 正確に言えば FORTRAN コンパイラーしかベクトル化してくれない. これは, C 言語でスーパーコンピューターを使う人 (使いたがる人) はあまりいないからで, C の言語仕様がスーパーコンピューター向きでないというわけではない (と思う).

[24] 行の終りに　`i = 1 "コメント　`のようにコメントを書くことのできる処理系もあるが (これは, HITACHI の VOS3 上

としてください．*compilation*バッファが開き，そこで make や cc などがサブプロセスで走り出します．ありがたいことにエラーメッセージはそのバッファに出力されます．

```
C-x `
```

を使えば，エラー行に飛ぶことができます．繰り返し使えば，すべてのエラー行にもいけます．

5.6　FORTRAN と C の比較

この節では，FORTRAN と C の使い勝手やどのように使い分けるべきかについて，2 つの観点から述べる．

1 点目は言語仕様の問題である．言語仕様の問題とは，例えば，フランス語の関係代名詞には性と数の区別があるので，関係代名詞節を多用しても文の明晰さが失われないで済むことも多いが，英語ではそうはいかない — といった，その言語そのものの問題である．

2 点目は言語の環境の問題である．これは，例えば，和英辞典にはいろいろ良いものがあるから英作文は楽だが，和仏辞典はろくなものがないから仏作文は大変だ — といった，言語を取り巻く環境の問題である．

どの言語をどのように使うかは，この二点をどちらも考慮して決める必要がある．

また，なにも FORTRAN と C の 2 言語のみを対立させて考えなくても良さそうでもあるが，我々の周りの UNIX 環境で使われているのは事実上この 2 つなのでこのような比較をしてみるのである．それと，観測や実験の分野ではこの 2 つ以外にも BASIC などが重要となることがあるが，ここでは，主に数値計算をする時のことを念頭において述べる．

5.6.1　FORTRAN と C の言語仕様の比較

「C は FORTRAN をほぼ含む」と言える．即ち，FORTRAN でできることのほとんど全ては C で楽にできるが，その逆は必ずしも真ではない[20]．

従って，C が使えるのなら FORTRAN を新たに学ぶのは簡単であるが，FORTRAN を使えても C を新たに学ぶのは簡単ではないかも知れない．

学ぶ，という観点から言うと，プログラミングの初心者には FORTRAN の方がとっつき易いであろう．C では pointer を避けて通ることはできないが，これが初心者には特に難関かも知れない．

C の言語仕様は FORTRAN と比べて一貫性があり綺麗である．柔軟性も高く，FORTRAN のようなゴツゴツした書き方から，エレガントな書き方，さらにはきわどく危険な書き方まで自由に選べる．その反面，良いプログラミングの仕方を身に付けていないと，読みにくく危険なプログラムを書いてしまう危険性も大きい．

主に使う言語はどちらにすべきかといえば，C を使いこなせれば言うことはないが，科学計算[21]程度であれば FORTRAN で十分であると言って良いだろう[22]．

ただし，これはあくまでも言語仕様にのみ注目した議論である．

5.6.2　FORTRAN と C の言語の環境の比較

言語の環境で重要なのは，自分の周囲の人がどちらの言語を使っているかということと，計算を支援する環境がどのくらい整っているかということである．

最初に，前者について述べる．

[20] 例外として目立つのは，FORTRAN では楽に記述できる 2 次元以上の整合配列が C では楽ではないことである．
[21] 微分方程式の数値解など
[22] 実験や観測でコンピューターを使う場合には状況によるし，BASIC などが必要なことも多い．

1 度 Makefile を作ってしまえば, いつでも最新のバージョンを作ることができます. make してもなにも起きなかった場合は, 最後にコンパイルして以来, ソースリストに修正がなかったということにもなります.

make は,C のコンパイル作業に限らず, いろいろな用途に使えます.

5.5.6　emacs の C mode

emacs は強力な editor と知られていますが, その理由の一つに幾つかのプログラミング言語用に作られたモードがあることが挙げられます. C も *emacs* によってモードが支援されている言語の一つです. 以下に述べることについてより詳しく知りたければ,*emacs* で Info を見るか, *GNU emacs manual* をお読みすることを薦めます.

字下げ

emacs の各言語のモードはどれも tab キーが字下げ関数を実行するように定義されます. この関数はその言語の字下げの慣習を知っていて, それに応じて現在行を字下げします. しかし, この字下げは自分の慣習に合っていないかも知れません. そのときは然るべき手段で字下げのルールを変更できます.

emacs に於ける字下げはかなり信用できるものなので, 括弧の自動対応と共に, 構文の間違いなどをプログラムを書いている時点で発見することが容易になります. つまり, 自分が予期したように *emacs* が字下げをしないのであれば, そのときは自分が予期したように構文を書いていない可能性が高いといえます.

略称

C や Lisp の変数名はとても長くなることが多くあります. この様なときに *emacs* の略称の展開はとても便利です. 特に動的な略称の展開は前もった準備も不要で, 非常に便利に使えます.

変数名を 2 回目以降書くときには, 他と区別できるまでの綴りを書いたらM-/ を押しましょう. すると, その綴りで始まる単語を, バッファのポイントの前からバッファを逆方向に検索して行きます. もし, 望む変数名でなければ M-/ を押すことを繰り返すとさらに逆方向に検索して展開してくれます. これにより綴りの間違いによるミスを事前に防ぐことができます.

tag table

幾つかのプログラミング言語はプログラムを分割してコンパイルする機構を備えています. それに対応して, 普通はプログラムを書いているファイルも分割して, 複数個のファイルで 1 つのプログラムを生成するスタイルを取ります. このとき問題になるのが, どの関数をどのファイルで定義したかの情報の管理です. この表を tag table と呼びます. この tag table を作る命令は *etags* と言います.tag table によって管理された情報を基に *emacs* はジャンプやファイルの自動読み込みなどを実行します.

COMPILE

C の compiler はエラーの位置を行番号で示します. また, 一度に沢山のエラーを出してくれるときもあります. 一方,*emacs* は行番号を表示する editor ではありません. もし,*emacs* と *kterm* とを使用してプログラムのコンパイル, 修正を繰り返すとしたらそれは大変な労力を要します.

emacs にはコンパイル (make) をして, エラー情報を集め, そこに容易に行ける様な機構があります. それが M-x compile です.

```
M-x compile
```

この例は, まだ工夫が足りません. 少し学習すると, 次のような手順を踏むようになります.

```
cc -c foo.c
cc -c sub1.c
cc -c sub2.c
cc -c sub3.c
cc -o foo foo.o sub1.o sub2.o sub3.o
```

-c オプションは, 最後のリンクフェイズ (まとめ上げる段階) の前でコンパイルを止め, オブジェクトファイル*.o を作ってくれるものです. これのどこがいいのかというと,4 つのソースリスト*.c のうち, 修正があったものだけを再コンパイルして, リンクし直す (5 行目) だけですむからです. 他の 3 つの (修正箇所のなかった) プログラムをわざわざ時間をかけてコンパイルしなくてもいいのです.

さて, これでいくぶん開発効率は上がりますが, まだ問題はあります. まず, ファイル名が複雑になったり, ソースの数が多くなったりした場合に, いちいちコマンドラインにファイル名を書き並べたり, いくつものオプション指定を書いたり修正したファイルのうちどれを再コンパイルするのかを書いたりすることを毎回正確にすることは人間にはできません. そこで我々は make を利用します.

make の機能はあまりにも多く, また強力なので, もっとも簡単な使い方だけを説明しましょう. テキストエディタでたとえば次のようなファイルを作ってください. ファイル名は Makefile とします. これは make が標準で使うファイル名です (オプションで別のファイル名も指定できます).

```
foo: foo.o sub1.o sub2.o sub3.o
        cc -o foo foo.o sub1.o sub2.o sub3.o
foo.o: foo.c myheader.h
        cc -c foo.c
sub1.o: sub1.c myheader.h
        cc -c sub1.c
sub2.o: sub2.c
        cc -c sub2.c
sub3.o: sub3.c myheader.h
        cc -c sub3.c
```

あとはコマンドラインから

```
%make
```

とするだけです. プロンプトに戻ってきたときは, foo ができているはずです.

これは,MS-DOS でいうところのバッチファイルと少し似ていますが, もっと強力です. 要するにこれは, ファイルの依存関係と最終更新日時を見て, 変更のあったファイルだけをコンパイルし直してくれるのです.

たとえば最初の 2 行を見てください. 「foo は foo.o,sub1.o,sub2.o および sub3.o でつくられているので, これらのうち foo より新しいファイルがあったら, 次のコンパイル命令を実行して foo を作り直せ」と読みます.

myheader.h のようなインクルードファイルを更新したような場合も, それをインクルードしているファイルをコンパイルしてくれます. myheader.h をインクルードしているファイルのところにその旨書いておくだけでいいのです (上の例では foo.o,sub1.o および sub3.o の後ろに書いてある). Makefile は, ファイルの依存関係を書いた行とコンパイル命令 (複数行にわたってもよい) の繰り返しで作ります. このとき, コンパイル命令の前には, 必ずタブを入れておいてください.

で調べてください.printf() の他にも,C の標準的なライブラリ関数は, このオンラインマニュアルで見ることができます.

また, g++ というオブジェクト指向の C もあります.

5.5.4 lint

C の文法は非常に自由度の高いものです. コンパイラのエラーチェックは, 本当に最低限のことしかしてくれません. そのため,C のプログラムではかなりきわどい書き方もできます. そして C のプログラムは, 簡単に暴走してしまいます. 別にきわどい書き方などしなくても, 本当に簡単に暴走します. C を使っている人なら 1 度や 2 度は, あるいはもっと,

```
Segmentation Fault(core dumped)
```

とか,

```
Bus Error(core dumped)
```

というメッセージにお目に掛かったことがあるはずです. C のプログラムが暴走する原因のほとんどは, ポインタがらみです.

また話が逸れてしまいました. チェックの甘いコンパイラに代わって, 比較的厳しい文法チェックをしてくれるのが lint なのです. ただし, lint は gcc 用ではありません.

lint の使い方は,

```
lint foo.c
```

のようにします. コンパイラを通るプログラムでも, おかしな書き方やバグの原因になりそうなことをしている部分があれば, 知らせてくれます.

リストを見たところは少しもおかしくないし, コンパイラもちゃんと通るのに, なぜか暴走する, そんなときに lint を使ってみてください. 思わぬバグを発見してくれることもたびたびです. 困ったときの lint 頼み, というところでしょうか.

値を代入しないうちに参照する可能性のある変数や, その関数の中で 1 度も使われていない変数などがあれば, それも知らせてくれるので, 恥ずかしいプログラムを書かずにすむという利点もあります. 人前に出すときには,1 度はそのプログラムを lint にかけるというのも手でしょう.

ただし,lint といえども万能ではありません.lint を通ったプログラムでも, 暴走するときは暴走するのだということは覚悟しておいてください.

5.5.5 make

make は別段 C 専用のユーティリティーというわけではありません.

作っているプログラムが小さなうちは, コマンドラインから

```
cc -o foo foo.c
```

としていても, それほどつらいものではありません. ところが, 世の中にはしばしばソースリストが何千行にもおよぶような巨大なプログラムが存在します. たった 1 行を訂正するたびにコンパイラを起動して数千行をコンパイルし直すのは, 無駄です. そこで, ソースリストをある程度まとまったモジュールに分割して別々にコンパイルしておき, 最終的な実行ファイルを生成するときにそれらをひとつにまとめ上げるという手段を使うことになります. この分割コンパイルは,C コンパイラを使う上での大きなメリットのひとつです. さきのライブラリにしても, その正体は分割コンパイルをしているのと同じことです.

```
cc -o foo foo.c sub1.c sub2.c sub3.c
```

場合には非常に有用な手段ですが, その際に上で述べたような文法の違いが思わぬ落とし穴として控えていることがあります.

どのコンパイラでも正常に動く書き方は存在します. それは保証します (ただし, それに縛られるあまり, 効率の悪いプログラムになることは十分あり得ます). 何回かコンパイラに怒られているうちに, どう書けばいいのかがわかってくることでしょう. そういう書き方をするように心掛けてください. そうすることで, 文法に対する理解が深まることもあります.

まあ, 処理系にべったりと依存した (たとえば X–Window 関連の) プログラムだったりすると, それほど神経質になる必要はありません. この場合は, その処理系の C でしか通用しない方言を使いまくってもそれほど問題にはなりません.

それよりも問題になることがあります. これは文法とは少し異なる問題なのですが, 意外とネックになることが多いのです. それは整数のサイズです. `int` といえば何ビットの整数を指すのか. UNIX では `int` といえば 32 ビット整数を指します. MS–DOS 上の C コンパイラではこれが 16 ビットなのです. もしも MS–DOS 上の C コンパイラで 32 ビット整数を使いたいのであれば, UNIX なら

```
int i;
```

とするところを

```
long i;
```

としなくてはなりません. ごく普通のプログラムなら, この差はどうということはないのですが, まれに (ものによっては頻繁に) 致命的なバグの原因になることがあります. 移植することを考えるのなら, どちらでも大丈夫だという場合 (ほとんどの場合はそうなのですが) を除いて, `short int`(または `short`) や `long int` (または `long`) と面倒でも書いたほうがいいと思います.

また, MS–DOS 上で使われている C コンパイラは 2 byte 文字, つまり漢字やかたかな, 平仮名を `printf`, `scanf` 等で使用しても問題がありません UNIX の上に存在する C は 2 byte 文字に対応していないことがあります. `printf` や `scanf` に 2 byte 文字を使うとエラーを引き起こすことがあります. おかしなことが起こった時は

```
man printf
```

などを見て下さい.

5.5.3　ライブラリ

C 言語では, サブルーチンのことを 「関数」と呼んでいます. 数学でいう, いわゆる関数 (引数を受け取って値を返すもの) とは少し異なり, function のもうひとつの意味である 「機能」といういい方のほうが, よりその性格をよくいい表しているような気がします. というのも,C のプログラムの関数は, 「返す値」よりも 「中で行う処理」に重点を置いていることが多いからです. C はこの関数の 「副作用」で仕事をしていることが多いのです.

話が少しそれましたが, ライブラリとは, この便利な関数の集合体です. 汎用的で利用価値の高い関数を集めて, どのプログラムからでも利用できるようにしているのです.

もっとも使用頻度の高いであろう関数, それは `printf()` でしょう. これは一応, 整数の値を返す関数なのですが, もはや 「関数」としての意味を失い, Pascal でいうところの 「手続き (procedure)」として機能しています. この `printf()` の使い方を知りたいときは,UNIX のコマンドと同様に

```
man printf
```

```
cc -o foo foo.c -lX
```

です.X のライブラリーが /usr/lib にない時には -L オプションを使用せねばならないかも知れません.

さて, 5.4.4 節でも述べましたが, geoph には cc の他にもうひとつの C コンパイラが存在しています. gcc です. gcc は GNU CC を略したものです[19]. gcc は cc の上位互換になっています.

gcc はシンタックスレベルでかなりの最適化をしてくれます. 特に RISC マシンでは人間がアセンブラで書くよりもよっぽどいいコードを出します.

gcc の使い方は cc とほぼ同じです. ただ, わざわざ gcc を使うのですから, -O という最適化オプションは最低つけておいてください. 実行ファイルの性能が格段に上がります.

```
gcc -O -o foo foo.c
```

-O オプションはもちろん cc にもありますが, gcc のように強力な最適化はできていないようです.

他にもコンパイルオプションはたくさんあります. ですが, 当面はこの程度でいいでしょう. もっと詳しく知りたい方は,

```
man cc
```

で調べてください.

5.5.2　他処理系から移ってくる方のために

C 言語の生い立ちはどの解説書を見ても載っています. いまさら語る気はありません. しかし, C 言語は生まれ育った UNIX の枠を超えて様々な処理系に移植されていますし, 困ったことにそれらの文法仕様は少しずつ違います. これは最初に C 言語がどういうものであるかということを書いた本である 「プログラミング言語 C」— カーニハン, リッチー, 1981, 通称 K&R, C 言語界におけるバイブル的な存在です— の書き方がいわゆる言語仕様の形を取っておらず, 少々曖昧で, コンパイラを制作する人の裁量に任されているためだといわれています. かくして, 同じ C なのに他の処理系に持っていくと動かないという事態が多発するようになってしまいました.

そこで, 統一した文法を作ろうという動きがあります. それが ANSI 規格です. 最近は K&R も, この ANSI 規格に対応した第 2 版 (カーニハン, リッチー, 1989) を出しています.

典型的な例を挙げましょう. 以下は関数 main() の宣言です.

```
main( argc, argv )
int  argc;
char *argv[];

void main( int argc, char *argv[] )
```

前者は UNIX の cc の書き方, ANSI 規格では後者の書き方も許しています. 後者は cc を通りません. 比較的新しい C コンパイラは ANSI 規格を導入しています. gcc は ANSI 規格に準拠しています. むろん従来の書き方でもだいじょうぶです. cc は UNIX 始まって以来の膨大なソースリストがあるわけですから, おいそれと妙な文法の変更をするわけにもいきません. その点 gcc は比較的融通が効きます.

自宅のパーソナルコンピュータでソースリストを入力し, 大まかなデバッグも済ませてから, 学校で実行するという使い方は, 利用時間と予算に制限のある教育用計算機センターでプログラムを制作しようとする

[19] GNU をご存じの方は多いでしょう. GNU とは, 例の GNU's Not Unix! (名前の定義のしかたが再帰的です) のことです. UNIX と似ていて UNIX より優れた環境を作ろうというのが GNU プロジェクトなのですが, gcc はその要となる C コンパイラです.

5.4.4　C 言語の古い規格と新しい規格

C 言語には古い規格と新しい規格とがあり, 現在はどちらもよく使用されています. もちろん, 少しずつ新しい規格の方が優勢になる傾向にはありますが, 古い規格が無視できるほどではありません.

長年に渡って, C の規格はカーニハン, リッチー, 1981(原著は 1978 年) に書かれているものでした (“第 1 版” と呼ぶことにします). これに対して, ANSI[18] は 1988 年に新しい規格を発表しました (“ANSI C” と呼ぶことにします.). これは, カーニハン, リッチー, 1989 に書かれています.

ANSI C のコンパイラーは, 第 1 版に従って書かれたプログラムの大部分をそのまま受け付けますが, その逆は成り立ちません.

言語仕様としては ANSI C の方が優れているので, そちらの方を積極的に使うことを勧めます. しかし, まだ ANSI C のコンパイラーが用意されていないシステムもあります.

第 1 版のコンパイラーのコマンド名は cc で, どんな UNIX システムにも用意されているはずです. 一方, ANSI C のコンパイラーとしては, gcc が geoph にはインストールされています.

5.5　少し進んだ C 言語の話題

5.5.1　C コンパイラの呼び出し方

UNIX 標準の C コンパイラは cc ですが, 5.4.4 節で述べましたように, cc は第 1 版の規格にしか対応していません.

テキストエディタで foo.c というプログラムを書いたとします. もっとも単純なコンパイルのしかたは

```
cc foo.c
```

です.

cc には豊富なコンパイルオプションがあります. オプションは- (マイナス記号) の後にひとつまたは複数の英数字を付けて, コマンドラインに並べて指定します.

さきほどの例では, 実行ファイルの名前がデフォルトの a.out となってしまい, 後で見たときになにをするプログラムだかわからなくなります. そこで, 実行ファイル名を指定するオプション-o [実行ファイル名] を指定します.

```
cc -o foo foo.c
```

こうすれば, コンパイラは a.out のかわりに foo という実行ファイルを作ってくれます.

数値演算ライブラリを有効にする-lm というものは覚えておく必要があります.sin や cos, それに log といった数値関数を使うにはこのライブラリが必要となります.

プログラムをコンパイルするときには,

```
cc -o foo foo.c -lm
```

という指定をする必要があります. ソースファイルに数値演算関係のヘッダファイルである math.h をインクルードする指定をしただけでは, つまり

```
#include <math.h>
```

としただけでは不十分です.

X 関係のグラフィックを使いたい時は同様に

[18] American National Standards Institute — この手の規格の標準に関しては権威がある.

printf("%lf\n", *variable*); は *variable* の値を文字列で表して標準出力に書き出します. %lf は, printf に対して *variable* が倍精度実数型であることを教えるものです. \n は出力の後改行しろ, という意味です.

プログラムの最初から順に簡単に説明します.

.....

#define NDAT 4　NDAT に 4 という定数値を与えるという宣言 (ここでは, 入力する数値の個数が 4 だから.)

.....

double dat, sum;　dat と sum は倍精度実数型の変数であるという宣言. (C では倍精度実数型を使うのが普通)

.....

for (i=0; i<NDAT; ++i){ ··· }
(1)i の値を最初 0 にし,
(2) 条件 i<NDAT が成り立つかどうか調べる.
(3) 成り立てば {···} の部分を実行し (4) に行く. 成り立たなければ for 文から抜ける.
(4)i の値を 1 増し (++i), (2) に行く.
即ち, この for 文は FORTRAN の do 文と同じように働く.

sum += dat;　sum = sum + dat; と同じ.

··· sum / (double)NDAT　NDAT は整数値なので, 倍精度実数型変数 sum を割る前に倍精度実数型に変換しておく.

実行する　コンパイルして出来た実行型ファイルの名前を, 例えば mean としましょう.
それをリダイレクションなしに実行すれば, キーボードから数値を読み込みディスプレイに結果を書き出します. この場合の実行例を挙げましょう.

```
geoph:/home2/s22516> mean
3.1
4
21.0
-0.3
6.95000
geoph:/home2/s22516>
```

最初の 4 個の数値はキーボードからの入力で, 最後の行は結果です.
入力する数値を先ほど作ったファイル *infile* から読み込む場合は

```
geoph:/home2/s22516> mean < infile
6.95000
geoph:/home2/s22516>
```

のようになります. さらに, 結果をまだ存在していないファイル *outfile* に書き込む場合は

```
geoph:/home2/s22516> mean < infile > outfile
geoph:/home2/s22516>
```

として下さい. 結果を見たかったら, cat などで *outfile* の中身を表示させて下さい.

結果の利用　計算の結果をグラフにしたい時などは, 上記のような方法で数値の並んでいるテキストファイルを作り, それを gnuplot などに入力してやると良いでしょう. 絵を書くソフトについては, 6 章などを参照して下さい.

のように改行して区切るか, または

```
3.1  4  21.0  -0.3
```

のように 1 個以上の空白で区切ったファイル *infile* をつくります. ここでは, 数値は 4 個です.

プログラムを書く　やはりエディターを使って,

```
#include <stdio.h>
#define  NDAT 4

void  main(void)  {

   double  dat, sum;
   int     i;

   sum = 0.0;
   for (i = 0; i < NDAT; ++i)  {
      scanf("%lf", &dat);
      sum += dat;
   }
   printf("%lf\n", sum / (double)NDAT );

}
```

のようなプログラムを書きます.

標準入力から値を読んでいるのは `scanf("%lf", &dat);` です. 最初の引数は書式を指定するもので, `%lf` は倍精度実数を意味します. 読み込むのは, 区切り記号 (空白や改行) が来るまでです. `&dat` は数値を格納する変数 `dat` のアドレスです. まとめると, `scanf("%lf", &dat);` は, 標準入力からはいってくる文字の並びのうち区切り文字で区切られた文字列の 1 つを倍精度実数を表すと解釈し, その値を `dat` に格納する, という働きをします.

> 変数」とはメモリーの (どこでもよいから) ある領域のことです. 変数に値を代入するとは, メモリーのその領域に値を書き込むことです. 変数を引数として関数 (例えば, `printf`) に渡してやるとは, その値のコピーを関数側で見ることが出来るようにすることです. `scanf(···, &dat)` のように, 関数の内部で `dat` の値を変える必要があるものに `dat` の値のコピーを渡してやっても意味がなく, メモリーのどこの領域の値を変えるべきなのかを教えてやらなくてはなりません. `&dat` は 「メモリーの中で `dat` という変数の領域はどこか」を示す値で, その変数 `dat` の “アドレス” と呼ばれます. その値を関数側で見ることが出来るように, アドレスの値を関数に渡してやるのです.
>
> それに対して, `printf` は引数の値を書き出す関数なので, 値のコピーを貰えば良いのです.
>
> まとめると,
>
> *function(variable);*
>
> は *function* に *variable* と名前がついたメモリーに書いてある値を渡すのに対して,
>
> *function(&variable);*
>
> は *function* に *variable* と名前がついたメモリーの位置を示す値を渡すのです. 後者の場合, *function* の中で *variable* の値を変更することが出来ます. *function* の中で, *variable* というメモリーに書いてある値を変更するには, *variable* というメモリーに書いてある値を知る必要があるのではなく, *variable* というメモリーがどこにあるかを知る必要があるのです.

5.4.2　コンパイルし実行する

コンパイルとは何かということについては 5.1.4 節を参照して頂くことにして, ここではそのやり方だけを説明します.

前節の例 example.c をコンパイルするためには, シェルのプロンプト (ここでは, geoph:/home2/s22516>) に対して

```
geoph:/home2/s22516> gcc -o example example.c
```

と入力して下さい[16]. プログラムに間違いがなければ, 実行型ファイル[17] example ができます.

これを実行するためには, プロンプトに対して

```
geoph:/home2/s22516> example
```

と入力します. すると,

```
geoph:/home2/s22516> example
hello, world
geoph:/home2/s22516>
```

となるはずです.

5.4.3　最も簡単な入出力

実際のプログラムに最も良くあるのは,

外から何かデータを読み込み,
それを元に何か計算をし,
結果を外に出す (見せる).

というタイプのものです. 入出力ができなくては何もできないに等しく, しかも, 問題の多い部分なので, この節では, 最も簡単な入出力の仕方を解説します.

以下で出てくるリダイレクションやテキストファイルといった概念については, 5.1.3 節を参照して下さい.

標準入出力を使ったプログラム　ここでは, テキストファイルからデータを読み込み, 結果をテキストファイルに書き込むプログラムの書き方を解説します. その際にリダイレクションを使い, プログラムの中でファイル名を指定することはしません. バイナリーファイルの使い方や, プログラム中でファイルを指定する方法については解説しません.

例として, テキストファイル *infile* に並んでいる数値を読み込んでその平均値を計算し, それを *outfile* に書き出す, というプログラムを取り上げます.

数値のファイルを作る　エディターを使って, 数値を

```
3.1
4
21.0
-0.3
```

[16] gcc は新しい規格に対応したコンパイラー.
[17] やはり, 5.1.4 節を参照して下さい.

2. **DIRECT ファイル**

DIRECT ファイルではレコードの長さを決めるので, レコードを認識するための記号などは付加されず, 決った長さのレコードがつながった最も単純なファイルとなる. (システムによってはファイルの先頭に固有のヘッダを付加するものもある.)
行末記号のようなものがないので, 変数の内部表現が同じであれば, ほとんどの計算機で互換性がとれる. さらに, システムによっては SEQUENTIAL ファイルに比べて, 処理速度が圧倒的に早い場合もある.
UNIX 系の計算機においては, たとえ単に順番にレコードを参照する場合でも, 利用価値のあるファイル形式である.

5.4　最低限の C 入門

この節では, まず, C を全く使ったことのない人を対象に,

プログラムを書き,
コンパイルし,
実行する

ということができるようになるための最低限の解説をします.

次に, 簡単な入出力を伴うプログラムの書き方を解説します.

前提とすることは, 何でも良いからかエディター (emacs など) を使い文書を編集しファイルに書き込むことができること, です.

C の文法や仕様については, 数多くの解説書が出ていますので[14] ここでは全く解説しません. ただし, 現在の段階では C 言語には古い規格と新しい規格の 2 種類が存在することに注意して下さい. このことについては, 5.4.4 節で説明します.

5.4.1　プログラムを書く

emacs などのエディターを使って C のプログラムを書き, ファイルにセーブします. ファイル名は .c で終わっていなくてはなりません.

ここでは, 次のようなプログラムを作ったとしましょう.

```
#include <stdio.h>

void  main(void) {
   printf("hello, world\n");
}
```

これは, 画面に hello, world という文字を書くプログラムで, 新しい規格にしたがった書き方をしています.

`#include <stdio.h>` は入出力の手続き `printf` などを使うという宣言. C のメインプログラムの名前は `main` でなくてはならい. C のプログラム単位は全て「関数」であるが, `main` の前の `void` によって `main` は値を返さない関数であることを宣言している. また, `main` の後の `void` は, `main` が引数を持たないことを示している. `\n` は画面に `"hello, world"` と書いた後改行することを指示する[15].

そして, このプログラムを書き込んだファイル名を `example.c` とします.

[14] 例えば, カーニハン, リッチー, 1989. これは, C を開発した人たちが書いている非常に良い教科書だが, 初心者には難しい. 初心者向けには, 林, 1991 が良い.
[15] これがなかったらどうなるか試してみなさい.

雑な構造をしているが, どんな長さのレコードも記録できる真の可変長レコード形式である (V, VB は可変長といってもレコード長の上限がある).

前述のように, 一般に計算機が読み書きする時には, できるだけ大きな単位にまとめて行う方が効率がよいので, ブロック化された形式の方が処理は早い.

メインフレームの計算機では, アクセス方法とファイルの構造は, ほとんど独立した概念として扱われる.

1. **SEQUENTIAL ファイル**

   書式なしの SEQUENTIAL ファイルでは, 通常 VBS 形式が使われる. この形式が最も FORTRAN 規格とは整合性がよく, DD 文などで何も指定しなければ, この形式が採用される場合が多い.
   書式つきの場合は, もともと, 特定の長さを持つデバイス (カード, 磁気テープなど) に出力することが多いので, F, FB なども使われる. この場合, プログラムが書き出すレコードの最大レコード長が, ファイルのレコード長を越えることがないように注意しなければならない.
   書式なしのデータを F, FB などに出力することは, FORTRAN の文法との整合性が悪いので, 禁止されているか, 制御コードを含む形式のファイルとして扱われることもある.
   どちらにしても, ユーザーがブロック長などを指定するということは, 単なる FORTRAN プログラマの知識を越えて, システム管理者的な知識が必要となり面倒である.
   しかし逆に言えば, プログラムで行う処理の性質とシステムの特性の両方を考慮して極限までチューンアップすることが可能である. 実際, メインフレーム系の計算機では, ブロック長などのパラメータの選び方でかなり処理速度に差が出る.

2. **DIRECT ファイル**

   DIRECT ファイルで使えるのは固定長レコードのファイルだけである. DIRECT ファイルはランダムにアクセスすることを可能にするため, ブロック化をしないことになっている. したがって, 順番にアクセスするならば SEQUENTIAL の方が早くなる.
   一般的に, メインフレーム系の計算機では DIRECT ファイルは扱いやすくないので, あまり使われていないようである.

- **UNIX 系**

UNIX 系 (もしくは MS-DOS 系, Mac) の計算機が扱うファイルは基本的に全て可変長レコードのファイルであり, 表向きブロック化という概念はない. 実際には, 何等かのブロック化が計算機内部で行われているものと思われるが, それは, FORTRAN ユーザーのレベルではわからない.

1. **SEQUENTIAL ファイル**

   書式つきの SEQUENTIAL ファイルでは通常のテキストファイルと同様にレコードの終りに行末記号が付加される. 行末記号は UNIX では 1 バイトで LF (line feed), MS-DOS では 2 バイトで LF (line feed), CR (carriage return) である.
   書式なしの場合には, レコードの最初と終り (または最初だけ) に, レコードを認識する何等かの記号が付加される. この記号は処理系によってかなりことなり, 同じ内部コードを採用している場合でも互換性はない.

まず, 書式のありなしの説明をする. 書式つきのファイルとは数字の内部表現 (5.3.1, 5.3.2 節) を文字に変換して記録しているものである. 書式なしのファイルとは, 内部表現をそのまま記録しているものである. つまり, 書式のありなしとは, いわば 「ファイルの中身」を指定するものだ. 内部表現は処理系に依存するので, 書式なしファイルは, 異なった処理系間では互換性がない. プログラム上では, 例えば,

```
1          WRITE(6,100) K
2      100 FORMAT(I4)
```

という具合に書いた時の出力が書式つきで, 一方,

```
1          WRITE(6) K
```

という具合に書いた時の出力が書式なしである.
次に, アクセス方法の指定の説明をする. SEQUENTIAL はファイルの頭から順番に読み書きするということで, DIRECT はファイルの中のレコードを直接指定して読み書きするということだ. プログラム上では, 例えば,

```
1          READ(5,100,REC=7) K
2      100 FORMAT(I4)
```

のようにレコード番号を直接指定して読み書きするのが, DIRECT で, そうでないのが SEQUENTIAL である.

実際には 「アクセス方法」よりも, そのアクセス方法を実現する 「ファイルの構造」の方が問題となる場合が多い. 「ファイルの構造」と 「アクセス方法」とは関係がある. このことを以下に説明していく.
規格上, OPEN 文において, DIRECT ファイルではレコード長を指定する RECL 指定子を書かなければならず, SEQUENTIAL ファイルの場合には, これを書いてはいけない. つまり, 「アクセス方法」の指定は, DIRECT ファイルは固定長レコードのファイルであり, SEQUENTIAL ファイルは可変長レコードのファイルであるという 「ファイル構造」を暗黙のうちに指定することになる. [13]
FORTRAN プログラム上の論理的なレコードが, 実際に記録媒体の上でどのように記録されるかということに関しても, 大きく分けてメインフレーム系 (IBM 系) と UNIX 系の 2 種類あり, それぞれファイルの扱い方がかなり異なる.

- メインフレーム系

  メインフレーム系の計算機では, ファイルの構造に関する概念にレコードの他にブロックというものがある. ブロックとは磁気テープや磁気ディスクなどに対する物理的な読み書きの単位である. 磁気テープなどにおいては, 普通いくつかのレコードをまとめたブロックという単位で読み書きが行なわれる. いちいちレコード毎に読み書きしていたのでは時間がかかり過ぎるからである.

  ファイルの構造としては, 固定長 (F), 可変長 (V), ブロック化固定長 (FB), ブロック化可変長 (VB), さらに, スパンドブロック化可変長 (VBS) などがあるが, これらの概念は FORTRAN にはないので, FORTRAN プログラム以外の部分 (DD 文など) で指定することになる.

  これらのファイル構造のうち, 固定長ファイル (F, FB) が最も単純で, 決った長さのデータが並んでいるだけで, 余分なコードは入っていない. F はブロック化されておらず (1 ブロック = 1 レコード), FB はブロック化されている. これに対して, スパンドブロック化可変長 (VBS) 形式は, 最も複

[13] ただし, 以下で説明するように, SEQUENTIAL かつ固定長レコードのファイルというのはある.

制御コード表

記号	意味	英語名	記号	意味	英語名
NUL	空値	null	DC1	装置制御 1	device control 1
SOH	ヘディング開始	start of heading	DC2	装置制御 2	device control 2
STX	テキスト開始	start of text	DC3	装置制御 3	device control 3
ETX	テキスト終了	end of text	DC4	装置制御 4	device control 4
EOT	伝送終了	end of transmission	NAK	否定応答	negative acknowledge
ENQ	問い合わせ	enquiry	SYN	同期信号	synchronous idle
ACK	肯定応答	acknowledge	ETB	伝送ブロック終結	end of transmission block
BEL	ベル	bell	CAN	取消	cancel
BS	後退	backspace	EM	媒体終端	end of medium
HT	水平タブ	horizontal tabulation	SUB	置換文字	substitute character
LF	改行	line feed	ESC	拡張	escape
VT	垂直タブ	horizontal tabulation	FS	ファイル分離文字	file separator
FF	改ページ	form feed	GS	グループ分離文字	group separator
CR	復帰	carriage return	RS	レコード分離文字	record separator
SO	シフトアウト	shift out	US	ユニット分離文字	unit separator
SI	シフトイン	shift in	SP	空白	space
DLE	伝送制御拡張	data link escape	DEL	消去	delete

富士通と日立の EBCDIC と ASCII の対応

コード	富士通	日立	ASCII
4A	c*	[	[
4F	|	!	!
5A	!	]	]
5B	¥	¥	$
5F	¬	^	^
6A		|	|
E0	$	$	\

- 注 1:富士通の 4A は c に縦棒.
- 注 2:ASCII コードとの対応は, 通常アスキー端末に出力されるコードで, これ以外のコードに変換される場合もある.

5.3.4　ファイルの構造

ファイルの形式も機種によりかなり異なる. [12]

FORTRAN においては, ファイルはレコードの集合体であるとみなされる. 1 レコードは 1 行のことだと思っておけば良い. この概念は, 昔, カードが使われていたころのイメージをひきずっており, レコードはカードの一枚一枚に相当する. 一枚のカードは 80 カラムあったので, 大体 80 字がレコードの標準的長さである.

FORTRAN 規格におけるファイルの属性は基本的に, 書式のありなし (FORMATTED/UNFORMATTED) とアクセス方法 (SEQUENTIAL/DIRECT) の組み合せで決る. 一応のお作法としては, OPEN 文中でこの属性を指定する. もっとも, 指定しなくてもプログラムは普通は走ってくれるから, そう気にする必要はないのかも知れない.

[12] 地球流体電脳倶楽部ライブラリでは, 固定長レコードのファイルに限って異機種間のデータ転送を可能にするためののユーティリティーとして FIOLIB を用意している.

ASCII コード表

	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	DEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

JIS X0201 では上の表の\が¥, ~が‾(上線) となっている.

EBCDIC コード表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE			SP	&	-						{	}	$	0
1	SOH	DC1					/		a	j	~		A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	DC3							c	l	t		C	L	T	3
4									d	m	u		D	M	U	4
5	HT		LF						e	n	v		E	N	V	5
6		BS	ETB						f	o	w		F	O	W	6
7	DEL		ESC	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z		I	R	Z	9
A					※	※	※	:								
B	VT				.	¥	,	#								
C	FF			DC4	<	*	%	@								
D	CR	GS	ENQ	NAK	(	)	_	’								
E	SO	RS	ACK		+	;	>	=								
F	SI	US	BEL	SUB	※	※	?	"								

- 制御コードの意味に関しては, この節の最後の表を参照のこと.
- 富士通のコードにはこれ以外の制御コードも定義されている.
- 日立の EBCDIK ではアルファベットの小文字のコードが異なる.
- 特殊文字部分の※ 印は富士通と日立で定義が異なる部分である.
- ¥(5B) は通貨記号であり, アメリカでは $ となる. その場合には, E0 の$は\ (バックスラッシュ) となる.

したがって, 小文字で FORTRAN プログラムを書くのは厳密に言えば文法違反になる.

これらの FORTRAN 文字を含めて, 文字の内部表現方法は FORTRAN の規格では特に規定されていない. 実数の内部表現方法と同様に, IBM 規格である EBCDIC と, アメリカの標準規格である ASCII とにわかれる.

- **EBCDIC**

  IBM によって定められた拡張 2 進化 10 進情報交換用コード (Extended Binary Coded Decimal Interchange Code). その名が示す通り, BCD(2 進化 10 進) コードと呼ばれるものを拡張したものである. BCD コードとは 16 進数の A,B,C,D,E,F を使わず, 0 から 9 までの数字を使って 10 進数を表現するもので, 4 ビットで 10 進数の 1 桁に対応させる.

  EBCDIC では 8 ビットで 1 文字を表現し, 数字は上位 4 ビットを F として下位 4 ビットを BCD コードに対応させている. (これが EBCDIC たる由縁) 富士通, 日立などのいわゆる IBM コンパチ汎用機でも採用されているが, 各社, 微妙に定義が異なる.

  FORTRAN 文字に関しては各社とも同じコードを使っているようであるが, その他の文字についてはかなり機種依存性が強い. 特に, 日本ではカタカナを使うためにコード体系を変更している場合があり, その対応の仕方に 2 種類ある. 一つは, アルファベットの小文字の部分をカタカナに置き換えてしまう方法で, この方法では, カタカナとアルファベットの小文字は同時に使用できない. 富士通の M シリーズではこの方法が採用されている (計算センターによって異なる場合がある). もう一つは, 小文字の部分を上記のようにカタカナに置き換えた上で, さらに, 小文字をあいているコードに押し込めたもので, EBCDIK(最後の K はカナ) コードとも呼ばれる. これは日立の M シリーズで採用されている. これら 2 つの方法で, アルファベットの大文字とカタカナに関しては同一のコードとなるが, 小文字に関しては互換性がなくなる.

- **ASCII コード**

  これはアメリカ規格協会 (ANSI) で規定された文字コード体系である. UNIX, MS-DOS などで採用されている. 日本では, ほぼ同じものが JIS X0201 として規定されている. ASCII コードは 7 ビットで, 最上位桁は 0 であるが, JIS には 8 ビット全部使ってカタカナまで規定した 8 単位符号表がある.

このように文字コードは, たとえ FORTRAN 文字であっても機種に依存する. したがって, 文字コードを返す `ICHAR` 関数は FORTRAN の標準の組み込み関数であるが, その値は機種に依存することになる. 文字コードの大小を機種に依存せずに比較するための関数として, `LGE`, `LGT`, `LLE`, `LLT` があり, これはその計算機のコード体系によらず, ASCII コード順に比較する.

であり, 絶対値最小のものは

$$2^{-1} \times 2^{-125} = 1.17549435 \times 10^{-38} \tag{5.8}$$

である. 非正規数 ($f_1 = 0$) まで含めて, 絶対値最小のものは,

$$2^{-24} \times 2^{-125} = 1.40129846 \times 10^{-45} \tag{5.9}$$

である.

また, この方法で表される正規数の相対誤差は $f_1 = f_2 = \cdots = f_{24} = 1$ の時, 最も小さくて

$$\text{約}\ 2^{-25} \approx 3 \times 10^{-8} \tag{5.10}$$

$f_1 = 1$, $f_2 = \cdots = f_6 = 0$ の時, 最も大きくて

$$\text{約}\ 2^{-24} \approx 6 \times 10^{-8} \tag{5.11}$$

である. ただし, これは実数を丸める時に 25 桁目を 0 捨 1 入 (四捨五入の 2 進数版) した場合で, IEEE 規格ではその他の丸めも許されている. 切捨ての場合は, この 2 倍になる. もちろん, 非正規数はこの限りではない.

このように, 同じビット数で実数を表現しても, 実数として表現できる範囲や精度はシステムによって異なる. 電脳ライブラリでは, このようなシステム依存の定数を MATH1/SYSLIB の GLPSET/GLPGET で管理している.

我々の身の回りにあるコンピュータで採用されている表現方法を以下の表に示す.

計算機	OS	コンパイラ	浮動小数点表現方法	備考
FACOM	MSP(汎用)	FORT77EX	IBM	
FACOM	XMP(UNIX)	FORT77EX	IBM	
HITAC	VOS3(汎用)	??	IBM	
HITAC	HIUXM(UNIX)	f77	IBM	
SUN	UNIX	SUN FORTRAN	IEEE	
PC9801	MS−DOS	F77L(Lahey)	IEEE	
PC9801	MS−DOS	BASIC	その他*	

注 (*): 2 進表現で IEEE に近いが, 符号ビットの位置や指数部のゲタの値などが異なる.

5.3.3　文字の内部表現

文字の内部表現も実数表現と同様に機種に依存する. [11]

FORTRAN では文法上, 以下の FORTRAN 文字集合が定められており, FORTRAN プログラムは, これらの文字だけで書かなければならない. (注釈行や文字型データの中は例外)

英字:　ABCDEFGHIJKLMNOPQRSTUVWXYZ
数字:　0123456789
特殊文字:　空白'()*+,-./:= 通貨記号

[11] 地球流体電脳倶楽部ライブラリでは, 機種依存する文字処理を規格化するためのユーティリティーとして CHGLIB, CHKLIB を用意している.

- **IBM 形式**

  これは富士通, 日立などのいわゆる IBM コンパチ汎用機で採用されている 16 進法の浮動小数点形式である. これらの計算機はもともと事務処理などのデータ処理が便利なように作られているため, 16 進演算が基本になっている. 各ビットの使い方は以下の通り.

0	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
±	$E+64$							f_1				f_2				f_3				f_4				f_5				f_6			

  ここで, 指数部は負にならないように Eに 64 を足した「ゲタばき表現」で表され, Eは-64 から 63 の値をとる. 仮数部は 16 進の 6 桁で, 7 桁目は切捨てられる. 0 は, この表示体系にはなじまない数値であるので, 別途,「全てのビットが 0 である数値」という定義がなされている.

  この方法で表される 0 でない数値のうち絶対値最大のものは

$$(1-16^{-6})\times 16^{63} = 7.237005\times 10^{75} \tag{5.3}$$

  であり, 絶対値最小のものは

$$16^{-1}\times 16^{-64} = 5.397605\times 10^{-79} \tag{5.4}$$

  である.

  また, この方法で表される数値の相対誤差は $f_1 = f_2 = \cdots = f_6 = 15$ の時, 最も小さくて

$$\text{約}\ 16^{-6} \approx 6\times 10^{-8} \tag{5.5}$$

  $f_1 = 1$, $f_2 = \cdots = f_6 = 0$ の時, 最も大きくて

$$\text{約}\ 16^{-5} \approx 10^{-6} \tag{5.6}$$

  である. このように f_1の値が小さいと, その上位ビットが無駄になってしまうので精度が極端に悪くなる.

- **IEEE 形式**

  これは UNIX マシンなどで採用されている 2 進法の浮動小数点形式である. 最近では, この形式を採用する計算機が多い. これは, IBM 形式に比べて相対精度が高いという特徴がある. 各ビットの使い方は以下の通り.

0	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
±	$E+126$								$f_2\cdot\cdot f_4$			$f_5\cdots f_8$				$f_9\cdots f_{12}$				$f_{13}\cdots f_{16}$				$f_{17}\cdots f_{20}$				$f_{21}\cdots f_{24}$			

  ここで, 指数部は Eに 126 を足した「ゲタばき表現」で表され, Eは-125 から 128 の値をとる. 仮数部は 2 進の 24 桁で, f_1は常に 1 と決っているので省略される. ただし, Eが-126 (ゲタばき表現の指数部が 0) の時には, $f_1 = 0$ と見なして, より小さな数を表現するが, 当然 2 進 24 桁の精度はない. また, $E = 129$ (ゲタばき表現の指数部が 255) の時は, 無限大など特殊な数字を表すのに用いられ, 実数演算はできない. 0 は全てのビットが 0 である.

  この方法で表される正規数 ($f_1 \neq 0$) のうち絶対値最大のものは

$$(1-2^{-24})\times 2^{128} = 3.40282347\times 10^{38} \tag{5.7}$$

32 ビットの符号つき整数の補数表示では, 0 及び正の整数は 「絶対値表示」同様に, 31 ビットの 2 進数の先頭に符号ビット (0) をつけたもので表される.
これに対して, 負の整数は, その整数の絶対値の 「補数」に符号ビット (1) をつけたもので表される. 一般に複数の 「補数」の定義があるが, ここで言う補数とは 「2 進数における 2 の補数」(真補数) のことである. この場合, ある数 Xの補数とは, 「Xに加えると 2^n になるような数」で, 「各ビットの 0 と 1 を入れ換え, 最下位桁に 1 を足した数」といっても同じである.
要するに, 符号つき整数を符号ビットまで含めて単純に 32 ビットの (正の)2 進数と見なした時, 負の整数 Nは $2^{31} + N$として表される (下表). この方法では, -2^{31} から $2^{31} - 1$ までの数を表現できる.

内部表現	数値
11111111 11111111 11111111 11111111	-1
11111111 11111111 11111111 11111110	-2
.....	.
.....	.
10000000 00000000 00000000 00000000	-2^{31}
01111111 11111111 11111111 11111111	$+2^{31} - 1$
.....	.
.....	.
00000000 00000000 00000000 00000001	1
00000000 00000000 00000000 00000000	0

この表現を用いると, 加減算において符号部分も含めて正の数と統一的に扱うことができるようになり, 演算回路の構成が簡単になるため, 多くの計算機でこの表現法が用いられている.

このように補数表示では, 内部表現のビット列と整数としての値が 1 対 1 に対応しているので, データを数字ではなくビット列として処理したい時には, 整数型の変数が使われる.

5.3.2　実数の内部表現

整数の内部表現に関しては, ほとんどの機種で同一の表現となっているのに対して, 実数型に関してはいくつかの規格が存在する.

一般に, 実数型の変数は浮動小数点の形で表現される. すなわち, β進法を使った場合, 実数は

$$\pm\,(0.f_1 f_2 f_3 \cdots f_m)_\beta \times \beta^{\pm E} \tag{5.1}$$

という形で表されている. ここで,

$$\pm\,(0.f_1 f_2 f_3 \cdots f_m) = \pm(f_1 \times \beta^{-1} + f_2 \times \beta^{-2} + f_3 \times \beta^{-3} + \cdots f_m \times \beta^{-m}) \tag{5.2}$$

は, 仮数部で f_iは 0 から$\beta - 1$ までの整数, $f_1 \neq 0$ である. また, Eは指数部で 0 または正の整数である.

ほとんどのメインフレーム系の計算機で使用されている実数表現は IBM 形式と呼ばれるものである. これに対して, UNIX などのコンパイラでは IEEE(アイ・トリプル・イーと読む) 規格を採用しているものが多い. どちらも, 32 ビットを 1 語とする点は同じであるが, 採用されている進法が異なり, 表現できる実数の範囲や精度に違いがある.

[10] 地球流体電脳倶楽部ライブラリでは, 実数表現形式の異なる計算機同志でデータを交換するためのユーティリティーとして REALLIB を用意している.

ぬ誤差を生じてしまう. しかも, 処理系によってこの誤差の生じかたが大きく異なるのでたちが悪い.

べき乗の精度が気になるところでは, その部分だけ倍精度演算をするなどの安全対策を心がけた方がよい. 特に, 基数 B の精度に気を付けること. たとえ, 1 つの計算機で所定の精度が得られても, 他の計算機で同じ結果が得られる保証はない.

- 定数の精度 :

  FORTRAN 規格では 「定数は, それを表す文字列そのものが型と値を決める」とされている. したがって, 定数の書き方によって値だけでなく, その精度も決められることになる. 例えば

```
1          DOUBLE PRECISION X
2          ....
3          X = 3.8*X*(1.-X)
4          ....
```

  と書くと, せっかく倍精度計算をしているのにも関わらず, 定数 3.8 や 1. が単精度の定数なので, 全体として倍精度計算の精度が出ない. 正確にいうと 3.8 という定数は X との乗算が行われる前に, 倍精度実数に変換されるのであるが, その倍精度実数の値はあくまで単精度の 3.8 であり, 倍精度の 3.8D0 とは値が異なる. このようなところでは, 必ず

```
1          X = 3.8D0*X*(1.D0-X)
```

  と書かなければならない. これは, DATA 文中の定数も同じである.

5.3　コードの内部表現と FORTRAN

本節では[9], 計算機の内部で数や文字がどのようなビット列で表されているのかということ (内部表現), また, それと関連して, FORTRAN におけるファイルの構造について取り扱う.

5.3.1　整数の内部表現

ビット操作をする変数には通常整数が使われる. その際に, 整数がどのようなビット列として表現されているかということが問題となる. また, ビット列を扱う場合には, 10 進の整数値として扱うより, 16 進数として扱った方が便利な場合もある. [10]

符号つきの整数を計算機の内部で表現する方法としては 「絶対値表示」と 「補数表示」がある. 「絶対値表示」とは整数の絶対値を 2 進数で表現して, その先頭ビットに符号ビット (正ならば 0, 負ならば 1) をつけたものである. この方法は素朴でわかりやすいが, 0 という整数の表現方法は 0 と -0 の 2 種類存在することになる.

「補数表示」は以下に示すように, もう少し込み入った定義になっているが, ほとんどの計算機では, この補数表示が採用されている.

- 補数表示

[9] 本節は地球流体電脳倶楽部ライブラリマニュアル 「MISC1」によるものである.
[10] 地球流体電脳倶楽部ライブラリでは ビット操作をするためのライブラリとして BITLIB を, 16 進数として扱った方が便利な場合のために HEXLIB を用意している.

例えば, 自分で外部関数を定義して, それを引用しているつもりでも, たまたまその処理系が同名の組み込み関数を持っていた場合には, ユーザー定義の関数ではなく, 処理系が用意した関数が呼ばれてしまう可能性がある. 関数名の長さは 6 文字以内と制限されている上, 人間の発想も多様性に限りがあるので, この手の衝突は意外に多い. しかも, この種の原因でプログラムが暴走したりすると, その原因を突き止めるのはかなり難しい. (自分のプログラムそのものには全く誤りがないのだから)

そこで, 処理系に依存しないプログラムを書くためには, そのプログラム単位で引用している外部関数名は全て EXTERNAL 文に指定することが望ましい. また, FORTRAN 規格にない処理系固有の関数を引用している場合には, INTRINSIC 文に指定すべきである.

なお, 当然のことながら, 電脳ライブラリが用意する関数は全て 「外部関数」である. 処理系から見れば電脳ライブラリを含めて全て 「ユーザープログラム」であるから, 自分が書かなかったからといって組み込み関数と混同することのないように注意されたい.

5.2.10　COMMON 文

前述の EQUIVALENCE 文が同じプログラムの中の変数を結合させるものであったのに対し, COMMON 文は異なったプログラム単位 (サブルーチンなど) の中の変数同志を結合させるものである.

初期の FORTRAN では引数によるデータの引渡しに時間がかかったため, COMMON 文によりメインプログラムからサブルーチンへデータを引き渡す方法が多用された歴史がある. しかし, 現在ではデータの引渡しに要する時間は, どちらの方法でもほとんど変らない. それより, COMMON 文を使うと, どのサブルーチンで値が代入されているのかわかりにくくなってしまうので, 可読性を高めるという見地から, COMMON 文によるサブルーチンへのデータ引渡はお勧めできない.

電脳ライブラリでも余程特殊な理由 (内部の作業領域確保など) がない限り, COMMON 文は使用していない. 各サブルーチンへのデータの引渡は全て引数を使っている.

5.2.11　その他

その他, FORTRAN でプログラムを書く場合の注意を以下に列挙する.

- 整数のオーバーフロー :

  ほとんどのシステムでは, 整変数がオーバーフローしても一切の警告を発する事なく, 上位の桁を切捨てて何事もなかったかのように計算を続ける.

  これは非常に危険な事であるが, 乱数を発生させるプログラムなどでは意識的にオーバーフローさせている場合もある.

  とにかく, 大きな整数を扱う場合には, 注意が必要である.

- 整数の切捨て及び剰余 :

  実数値を整変数に代入すると, 当然小数点以下は切捨てられるが, その切捨ては常に 「0 に近くなるように」切捨てられる. これは, 正の範囲では数学的な切捨てと同じであるが, 負の範囲では異なる. (-1.5 が切捨てられると -1 になってしまう) また同様に, 組み込み関数の MOD も負の範囲で数学的な剰余の定義と異なる.

- 実数のべき乗 :

  べき乗計算 X = B**N では, 基数 B の丸め誤差が N 倍になって現れるので, 丸め誤差だけでも N が大きいとかなりの誤差を生じる. さらに, 計算の過程で生じる誤差が加わると, 思わ

という様に使うとサブルーチン SUM は, 以下のように 2 次元配列要素 Y(2,1) を先頭とする記憶領域を 1 次元配列 X に割り当てる.

Y(2,1)	Y(3,1)	···	Y(10,1)	Y(1,2)	Y(2,2)	···	Y(10,2)	Y(1,3)	Y(2,3)	···
X(1)	X(2)	···	X(9)	X(10)	X(11)	···	X(19)	X(20)	X(21)	···

その 1 次元配列 X の要素を 10 (JX) 個おきに 10 個の和をとると, Y(2,1), Y(2,2), ···, Y(2,10) という具合に, 二次元配列 Y の第 2 添字についての和をとることになる.

ここで, サブルーチンの中では配列 X の長さはわからず, 単に, X(1) を先頭とする 1 つながりのデータとして扱われていることに注意して欲しい. 配列がサブルーチンに渡されるとき, メインプログラムの配列の中身がサブルーチンの配列の中身にコピーされるのではなく, メインプログラムの中の配列の番地だけが渡され, サブルーチンの方はその番地を基準に処理を行うのである. このようなことを意識してプログラムを書くと, 少ない引数で柔軟な処理が可能になる.

もう一つの応用例を紹介しよう.

```
1          SUBROUTINE SUM2(X, IMAX, IX, JX, XSUM)
2          REAL X(IMAX,JX)
3
4          XSUM = 0.
5          DO 100 I=1,IX
6          DO 100 J=1,JY
7            XSUM = XSUM + X(I,J)
8      100 CONTINUE
9          END
```

これは 2 次元配列 X(I,J) の一部の和をとるものであるが, これを

```
1          REAL X(102,102)
2          ........
3          CALL SUM2(X(2,2), 102, 100, 100, XSUM)
4          ........
```

という具合に使うと, 102×102 の 2 次元配列のデータのうち, 周辺を残して真中の 100×100 の部分, すなわち, (2,2)–(101,101) を対角線とする部分の和をとることができる.

なお, サブルーチンに渡すことのできる引数のなかに, 「手続き名」がある. これは, 他の引数と異なり「変数」ではないので, 動的な変更はできないが, これが指定できるおかげで, 例えば地球流体電脳ライブラリの MATH1 の VRFNA/VIFNA の様に, サブルーチンを呼ぶ際に使うべき関数を指定することが可能になる. この場合, 実引数に指定する手続き名は, サブルーチンを呼ぶプログラムの中で EXTERNAL 文または INTRINSIC 文により手続き名であることを宣言しておかなければならない.

5.2.9　EXTERNAL 文と INTRINSIC 文

EXTERNAL 文は, そこに指定された関数名が外部関数であることを宣言するものであり, INTRINSIC 文は, そこに指定された関数名が組み込み関数であることを宣言するものである. これらは, 上記のように関数名をサブルーチンの実引数として使用する時に必ず必要なものであるが, それ以外も重要な意味がある.

SIN や COS のような組み込み関数は, FORTRAN 規格によって処理系が用意しなければならないが, FORTRAN 規格はさらに処理系が規格外の組み込み関数を用意することを許している. したがって, 各処理系は FORTRAN 規格に定められた組み込み関数よりも多くの組み込み関数を持っているのが普通である. そして当然のことながら, それら拡張された組み込み関数の名前に関しては, 処理系によって全く異なる. ということは, FORTRAN 規格にない関数名で, EXTERNAL 文や INTRINSIC 文によって宣言されていない関数名は, 処理系によって組み込み関数として解釈されたり外部関数として解釈されたりすることがある.

5.2.7　多次元配列の記憶順序

FORTRAN77 では多次元 (7 次元まで) の配列を定義できるが, 実際の記憶装置は多次元構造を持っているわけではなく, 1 次元的に管理されている. 多次元配列を 1 次元的に展開する順序は, FORTRAN77 の規格で定めてあるので, 多次元配列を 1 次元配列と結合する (多次元配列に 1 次元配列の別名を与える) ことが可能である. 例えば

```
1         REAL          X(6), Y(2,3)
2         COMPLEX       Z(3)
3         EQUIVALENCE (X,Y,Z)
```

という場合, 変数 X, Y, Z はどれも長さが 6 語 (48 バイト) の配列で, EQUIVALENCE 文により同じ記憶領域を占める. この時, それぞれの変数の並び方は

X(1)	X(2)	X(3)	X(4)	X(5)	X(6)
Y(1,1)	Y(2,1)	Y(1,2)	Y(2,2)	Y(1,3)	Y(2,3)
Re(Z(1))	Im(Z(1))	Re(Z(2))	Im(Z(2))	Re(Z(3))	Im(Z(3))

という様に, 2 次元配列 Y は添字の左側の数字から変るように 1 次元的に展開される. また, 複素数データ Z は 2 つの実数が並んだ形で展開される. したがって, X(3) の数値は Y(1,2), Re(Z(2)) と全く同じになる.

この規則は多くのプログラムで積極的に使われており, 多次元配列を 1 次元配列として扱ったり, 複素数データを実数データを見なして処理したりすることが行われている. この規則は, 計算機のハードウェアに近い部分の規則なので, 計算機によって異なるように思えるかもしれないが, FORTRAN77 規格で定められた「標準語」である.

5.2.8　サブルーチンへのデータ引渡

引数によるデータの引渡では, 前述の配列の記憶順序を意識したテクニックがよく使われる. 地球流体電脳ライブラリを始めとして通常の FORTRAN のライブラリパッケージにのサブルーチンの引数には 1 次元配列の要素を飛び飛びに使うための引数 (JX など) が必ずある. これは, 多次元配列を同じサブルーチンまたは関数で処理するためのものである. これらのルーチンの動作を理解するために, 以下のプログラムを見て欲しい.

```
1         REAL X(100)
2         ......
3         CALL SUM(X, 100, 1, XSUM)
4         ......
5         END
6
7         SUBROUTINE SUM(X, N, JX, XSUM)
8         REAL X(*)
9
10        XSUM = 0.
11        DO 100 I=1, JX*(N-1)+1, JX
12          XSUM = XSUM + X(I)
13    100 CONTINUE
14        END
```

このサブルーチンは 1 次元配列 X の和を求めて, XSUM として返すものである. この同じサブルーチンを 2 次元配列 Y に対して

```
1         REAL Y(10,10)
2         ........
3         CALL SUM(Y(2,1), 10, 10, YSUM)
4         ......
5         END
```

実際の観測データなどを扱う分野では, 文字と数字が混在したデータを扱うことも少なくないので, 伝統的にこのような EQUIVALENCE 文を使っている場合もあるが, 実際にこのような結合ができないコンパイラもあるので, 注意が必要である.

さらに, 文字型変数は他の型の変数と違って, 以下の例のように, 同じ文字要素を式の右辺と左辺に書く (3 行目) ことが禁じられている.

```
1         CHARACTER*12 CX
2         CX = 'DENNOU'
3         CX = CX(1:6)//'DENNOU'
```

文字以外の変数では, 同様のことが許されているので, つい, このような代入文を書いてしまいそうになるが, このような場合には,

```
1         CHARACTER*12 CX
2         CX = 'DENNOU'
3         CX(7:12) = 'DENNOU'
```

とすれば, 問題はない.

5.2.6　フォーマットの動的変更

FORMAT 文を用いて WRITE 文などで出力する書式を指定する場合, 書式の中に変数が使えないため, 「プログラムの中で必要な桁数を判断して, 出力する桁数を変える」という様な芸当ができない. このあたりが, 古い言語の堅さの 1 つなのであるが,　「だから FORTRAN はきらいだ」という前に, 以下の説明を読んで欲しい.

FORTRAN77 では WRITE 文の書式識別子に FORMAT 文の文番号ではなく, 以下のように, 文字変数を指定することができる. この機能を使えば, 書式の動的な変更が可能になる.

```
1         CHARACTER CFMT*12
2         ...
3         CFMT = '(T12, F5.2)'
4         WRITE(6,CFMT) X
5         ...
```

この例では, 文字変数 CFMT に定数を代入しているだけなので, 動的に書式を変更するようにはなっていないが, CFMT は変数であるから, プログラムの中で変更するのは容易である.

桁数などの数値変数を文字変数に書き込むには, やはり WRITE 文を使う. WRITE 文の出力機番を書くところに文字変数を書くと, その文字変数を “内部ファイル” として, 普通の WRITE 文と同じ様に編集, 出力ができる.

```
1         CHARACTER CFMT*12
2         ...
3         CFMT = '(T12, Fx.2)'
4         WRITE(CFMT(8:8),'(I1)') N
5         WRITE(6,CFMT) X
6         ...
```

この例では, 整変数 N を文字変数 CFMT の 8 桁目に書き込み, X を出力する際の桁数としている. このあたりのテクニックは FORTRAN66 の時代には使えなかったので知らない人も多いが, 知っていると非常に便利である.

この方法が前の 5.2.3節の方法 (各プログラムが IMISS 設定用の ENTRY を持つ方法) より優れている点は, GLPGET が持つ情報を複数のサブルーチンから参照できることである. 例えば, 欠損値を示す値は何も初期値としては 999 が与えられているが, 999 がデータ範囲に入っているような場合には, これを変更しておかなければならない. そのような時にでも GLPSET を一度呼ぶだけで, 欠損値処理をする全てのサブルーチンの動作を制御することが可能になる.

なお, GLPGET/GLPSET では EQUIVALENCE 文を使って実変数, 整変数, 論理変数の 3 つの型を 1 つのプログラムで扱っている. これは, 前述の通り, 厳密にいえば文法違反である. 例えば

```
1          CALL GLPGET('RMISS', RMISS)
```

という文を実行すると, 実引数 RMISS はサブルーチンの仮引数 IPARA と結合される. これは, サブルーチンの実引数と仮引数は型が同じでなければならない, という FORTRAN77 の文法に違反する. さらに, 26 行目で IPARA に IX(14) を代入するが, IX(14) は定義されておらず, その中身は EQUIVALENCE 文で結合された RX(14) であるので, 整数値としては 「不定」である. したがって 「不定」の値を代入する 26 行目も文法違反ということになる.

しかしながら, IPARA という整変数は, 単に値を RX(14) から RMISS に転送するためだけに使われ, その中身は常に実数値としての意味を持っているので, 実際には, まず, 問題を起さない. というより, 1 語長のビット (32 ビット) 全てを使って整変数を表現する普通の計算機では, どんなビットの組み合せでも整数値として解釈できてしまうので, IPARA の中身が整数として定義されていないこと, すなわち 「不定」であることを検出しようがない. つまり, この文法違反を見抜くことはできないはずなのである.

と言うわけで, これはほとんど完全犯罪なのであるが, 文法違反であることには変りはないので, この真似をすることは, あまりお勧めできない.

5.2.5　文字型変数とその他の変数の違い

前節では実変数, 整変数, 論理変数を混同した書き方をしても, 大きな問題にならなかった. これは, FORTRAN 規格により, これら 3 つの型のデータが全て 「1 数値記憶単位を占める」と規定されているからである. これに対して, 文字型のデータは 1 文字が 「1 文字記憶単位を占める」と規定されている. 多くの処理系においては 「1 数値記憶単位」は 32 ビットの 「1 ワード」に対応し, 「1 文字記憶単位」は 8 ビットの 「1 バイト」に対応するが, 規格の中で具体的な長さは規定されていない. したがって, 文字型とその他の変数を EQUIVALENCE 文で結合したりすると, 規格上, 「数値記憶単位」が 「文字記憶単位」のいくつ分に相当するかわからないため, そのような結合は禁じられている.

さらに, これは規格上の話だけではなく, 上記のような 1 ワードが 4 バイトに相当する標準的な処理系で, ユーザーが 「数値記憶単位」と 「文字記憶単位」の対応をよく考えた上で結合したとしても実害が生じる場合がある. これは, 文字型変数が文字単位で任意の長さがとれることから, 文字型変数の扱い方が, それ以外の変数 (実数, 整数, 論理変数) と大きく異なる場合があるからである. 特にサブルーチンの引数の渡し方が異なる場合が多い.

もともと FORTRAN66 の規格には文字型変数はなく, 文字を扱う場合にはキャラクターコード列 (整数) として扱われていた (4 文字を 1 つの整数として扱っていた). そのため 4 文字単位の長さの文字列しか扱えず不便であったが, FORTRAN77 から文字型変数として正式に認知されるようになった. これに伴って, 任意の長さの文字列が扱えるようになるなど, 文字の扱いが非常に楽になった反面, 上記のような制約も増えた. もっとも, かなり多くのコンパイラでは文字型とそれ以外の変数の結合が可能であるが, これも 「方言」の一つである.

に IOU0 を引用することはできないので, IOUSET が呼ばれたときに IOU に代入しておかなければならない. また, このようなプログラムでも SAVE 文と DATA 文は不可欠である.

5.2.4 地球流体電脳ライブラリ GLPSET/GLPGET の構造

地球流体電脳ライブラリの「パラメータ掲示版」である SYSLIB の GLPSET/GLPGET は, 毎回指定する必要のないパラメータをうまく隠蔽して見えないようにしておくという前述のテクニックを極限にまで利用したライブラリなのである. GLPSET/GLPGET は他のプログラムにパラメータを渡すだけで, あとは何もしない. その構造は, 以下のようになっている (実際のソースより簡略化されている).

```
1   *---------------------------------------------------------------------
2   *     GLPGET / GLPSET
3   *---------------------------------------------------------------------
4         SUBROUTINE GLPGET(CP,IPARA)
5
6         CHARACTER CP*(*)
7         PARAMETER (NPARA=17)
8
9         INTEGER   IX(NPARA)
10        REAL      RX(NPARA)
11        LOGICAL   LX(NPARA)
12        CHARACTER CPARA(NPARA)*8
13
14        EQUIVALENCE (IX,RX,LX)
15        SAVE
16
17        DATA CPARA( 1)/'NBITSPW '/, IX( 1)/32/
18        ........
19        DATA CPARA(12)/'LMISS   '/, LX(12)/.FALSE./
20        DATA CPARA(13)/'IMISS   '/, IX(13)/999/
21        DATA CPARA(14)/'RMISS   '/, RX(14)/999.0/
22        ........
23
24        DO 10 N=1,NPARA
25          IF (CP.EQ.CPARA(N)) THEN
26            IPARA=IX(N)
27            RETURN
28          END IF
29     10 CONTINUE
30        エラー処理
31        STOP
32  *---------------------------------------------------------------------
33        ENTRY GLPSET(CP,IPARA)
34
35        DO 15 N=1,NPARA
36          IF (CP.EQ.CPARA(N)) THEN
37            IX(N)=IPARA
38            RETURN
39          END IF
40     15 CONTINUE
41        エラー処理
42        STOP
43        END
```

ここで, CPARA という文字変数にパラメータ名が登録されており, その名前に対応した値が IX, RX, LX という変数に入っている. (これら 3 つの変数は EQUIVALENCE 文により結合されているので, 実際には 1 つの記憶領域しか持たない)

さて, 値を参照するためには次のようにすれば良い. 上の例でいえば欠損値 IMISS の値をとってくるためには

```
1         CALL GLPGET('IMISS', IMISS)
```

という文を実行する. この時, GLPGET では, CPARA の中から'IMISS' という名前を探して (24–29 行目) その名前に対応する値 IMISS として返す. この時返される値は, DATA 文によって指定されているが, 必要であれば, あらかじめこれらの値を GLPSET で変更しておくこともできる.

異なる型の変数が EQUIVALENCE 文などにより結合しており, その一方の変数が確定したとき, もう一方の変数は不定となる. 例えば,

```
1         INTEGER IX
2         REAL    RX
3         EQUIVALENCE (IX, RX)
4
5         RX = 1.
6         WRITE(6,*) IX
7         END
```

というプログラムで, 実変数 RX が代入されると, 整変数 IX の値は不定となる. したがって, 不定の変数を引用している WRITE 文は, 文法違反である.

しかし, この時 IX は, ちゃんと 「ある値」を持っている. ただし, 整数や実数の表現形式がコンパイラによって異なるため, この 「ある値」が, どのような値になるか FORTRAN 規格からは予測できないという意味で 「不定」なのである.

したがって, 文法上このプログラムの結果はわからないが, 通常は正常に動作して 「ある値」を書き出すことになる.

この種の文法違反は, しばしば意識的に行われることがあり, 地球流体電脳ライブラリでも, パラメータを管理するルーチン (GLPGET/GLPSET) などで, 意識的に行われている. (5.2.4 節参照.)

5.2.3 ENTRY 文

FORTRAN でサブルーチンを書く時, いろいろな場合を想定して, なるべく汎用的なサブルーチンにしようとすると, 引数がどんどん増えて困ることがある. 大事な引数は 2–3 個なのに, どうでもよいような引数がずらずらと並んでいると, それだけで CALL .... と書く手が重くなってくる.

そのような時に ENTRY 文を使うと解決できる場合がある. ENTRY 文とは, サブルーチンの途中から実行を始めるための文である.

```
1   *----------------- main program ----------------
2         REAL X(10)
3         .......
4         CALL TABLE(X, 10)
5         .......
6         END
7   *-----------------------------------------------
8         SUBROUTINE TABLE(X,N)
9         REAL       X(N)
10        DATA       IOU /6/
11        SAVE
12
13        WRITE(IOU,'(10F12.4)') X
14        RETURN
15
16        ENTRY IOUSET(IOU0)
17        IOU = IOU0
18        RETURN
19        END
```

このサブルーチン TABLE は 1 次元配列 X を出力するものであるが, TABLE だけを呼べば, DATA 文で指定された IOU の機番 (6) に出力される. ファイルなど別の機番に出力させたい場合は, TABLE を呼ぶ前に IOUSET を呼んで, IOU を書き換えておけばよい. つまり, 1 つのサブルーチンに, 印刷すべき変数だけを与える入口と, その他のパラメータを設定する入口の, 2 つの入口を作るわけである.

ここで, 注意すべきことは SUBROUTINE 文または ENTRY 文で指定された引数は, その SUBROUTINE 文または ENTRY 文から入った時でないと使えないことである. 例えば, このプログラムの ENTRY 文から入った時 (IOUSET が呼ばれた時), 変数 N を引用することはできない. 同じように, TABLE が呼ばれた時

これは, 最もわかり易い　不定」の場合である. 変数に値が代入なければ, 当然その変数にどのような値が入っているかわからない.

コンパイラによっては全ての変数を初期化 (通常は 0 を代入) するものもある. FORTRAN 規格はあくまで　不定」である (どうしてもよい) ので, そのようなコンパイラでも規格に違反しているわけではない. しかし, これは規格によって保証されているものではなく, 一種の　方言」であるので, これに頼ってプログラムを書くと後で必ず後悔する.

最初から変数に値を入れておきたい時には DATA 文を使う. 次の例を参照のこと.

- サブルーチンの中の **RETURN** 文または **END** 文を実行したとき:

  サブルーチンの中で使われる変数は, いくつかの例外を除いて, サブルーチンの実行が終了した段階で　不定」となる.

  これは少々わかりにくいが, 要するに, 同じサブルーチンを 2 回呼んだとき, 1 回目で値が代入された変数を, 2 回目に参照しようとしても, 1 回目の値が保存されている保証はない, ということである.

  例えば, 次のプログラムを見て欲しい.

```
1   *------------------ main program ----------------
2         DO 100 I=1, 100
3           CALL BEAT
4     100 CONTINUE
5         END
6   *------------------------------------------------
7         SUBROUTINE BEAT
8         DATA N /0/
9         SAVE
10
11        N=N+1
12        IF(N.EQ.10) THEN
13          WRITE(6,*) ' I hate you !'
14          N=0
15        ENDIF
16        END
```

  このサブルーチン BEAT は呼ばれた回数 (N) を覚えていて, 10 回呼ばれるごとにメッセージを出力するものである.

  変数 N は BEAT が呼ばれる前に, DATA 文によって 0 という値で確定している. BEAT が 1 回呼ばれると, この変数 N に 1 が加えられるが, もし, 9 行目の SAVE 文がなければ, 16 行目の END 文を実行した段階で, FORTRAN の規定により N は不定となる.

  SAVE 文は, サブルーチン内の変数の値をサブルーチンの実行が終っても保持するように指示する宣言文である. このプログラムのように変数 N の値が次回の呼び出しまで保持されないと正常に動かないようなプログラムでは必ず書かなければならない.

  コンパイラによっては (というより多くのコンパイラでは) サブルーチンの中で使われる局所変数を保存するが, これも　方言」である.

  ただし, DATA 文で指定された変数は, その値が書き換えられない限り, RETURN 文または END 文を実行しても　不定」にはならない.

  因みに, 地球流体電脳ライブラリの GLPGET/GLPSET が, 掲示板の役目を果たせるのは, この DATA 文と SAVE 文のおかげである. (5.2.4節参照.)

  なお, SAVE 文を指定すると, 通常, 実行ファイルのサイズは大きくなるが, 実行速度は若干速くなることが多い.

- 異なる型の変数が結合しているとき:

たプログラムは, たとえ, FORTRAN77 規格のコンパイラでコンパイルできたとしても, FORTRAN77 規格準拠のプログラムではない.

そこで, FORTRAN で何とかプログラムが書けるようになった初心者も, この道 10 年のベテランも, 市販の文法書以外に 『岩波 FORTRAN 辞典』(西村ほか, 1986) を座右におかれることを強くお勧めする. これは単なる文法書ではなく, 規格が意味するところや, 歴史的背景などが解説してあって, 辞典としてだけでなく, 読物としても面白い書物である.

プログラムを書く上での一般的な注意としては, 極力この 『方言』を避ける, ということができる. どうしても 『方言』を使わなければならないようなところは, 直接 『方言』を使わず一枚上にかぶせ, どこが方言であるのかを明解にわかるように構造化しておくのが良い. こういう構造をソケットという. ソケットから上のレベルのプログラムは方言に依存しないことを保証するわけだ. かくして移植のしやすさ (可搬性) が確保されたプログラムが得られる. FORTRAN の文法をよく理解して, 不必要な 『方言』を使わないよう心がけて欲しい.

以下の節では不必要な方言を避けるために知っておくべき知識, 知っていると便利なことがら, そして, プログラミング上の盲点となるような落し穴について多少の解説をしておく.

5.2.2　不定の概念

FORTRAN 規格の基本的な概念に 『不定』という概念がある. これは最も重要な概念の一つでありながら, 最もわかりにくい概念でもある. FORTRAN を勉強したことのある人ならば, 文法書の中の

○ ○ の時, 変数× × は不定となる.

という記述を見て, 首をかしげた経験が 1 度はあると思う. 最初にこれを読むと, 『× × に乱数が代入される』とか 『数値がメモリ上から消滅する』ということを想像してしまうが, 決してそういう意味ではない. 大抵の場合, ○ ○ の時, × × がどうなるか, コンパイラはちゃんと知っている.

この奇怪な文章を理解するには, FORTRAN の規格の文章が誰に対する文章であるかということを, よく理解しておく必要がある. FORTRAN 規格というのは, 我々プログラマに対する文章であると同時に, FORTRAN コンパイラをつくる人に対する文章でもある. したがって, この文章は 2 通りに翻訳できるように書いてあるのである. 上記の文章を, コンパイラをつくる人に対する文章に翻訳すれば,

○ ○ の時, 変数× × が占めていた記憶領域をどのように使ってもよい.

ということになり, 我々プログラマに対しては,

コンパイラをつくる人に, 上のように言ってしまったので, × × がどうなるかわからない.

ということになる.

もし, この文章をコンパイラを 『作った人』から 『使う人』へのメッセージだと解釈してしまうと, 『自分で作っておきながら, わからない, どういうことだ』と言いたくなってしまう. 特に, コンパイラに付属の文法書を読んでいると, そのような錯覚に落ちいりやすいが, FORTRAN の文法書とはあくまでも,

コンパイラを作る人と, それを使う人の両者の責任関係を明確にするための法律

なのである.

『不定』とは逆に, 規格によって値が保証されている状態を 『確定』という. 規格上, プログラムの中で引用できる変数は 『確定』された変数だけである. 以下, 変数が 『不定』となる例である.

- 変数が未定義のとき:

いう印象も免れない. しかしながら, FORTRAN には科学計算用の言語として, 他の言語にはない使いやすさがある. 単に 「蓄積が大きい」というだけでなく, この 「使いやすさ」こそ, FORTRAN が 30 年以上も使い続けられてきた真の理由である.

FORTRAN に限らず, どんな言語でもそれを使いこなすのは, なかなか大変である. 幸にも, FORTRAN の仕様は比較的小さい (文法書が薄い) ので学習は容易なほうであるが, それでも FORTRAN を “極める” のは容易なことではない. ここにでは入門書には書いてない 「奥義」をいくつかあげておくことにする. 詳細は 「地球流体電脳ライブラリマニュアル」を参照されたい[8].

5.2.1　FORTRAN の方言

計算機言語には各国の標準機関 (日本では JIS) 及び国際標準機関 (ISO) により定められた規格がある. FORTRAN も例外ではなく,

- **アメリカ規格:**　ANSI X3.9-1978 Programming Language FORTRAN
- **国際規格:**　IS 1539 Programming Languages — FORTRAN
- **日本工業規格:**　JIS C 6201 電子計算機プログラム言語 FORTRAN

という言語規格があるが, 実質的には全て同じもので, いわゆる FORTRAN77 の規格である.

言語規格というのは 「日本語の文法」の様なもので, 計算機に自分が意図した動作をさせるためにはこれをよく知っていなければならない. これは, 当り前のようで, 意外にくせものである. なぜならば, 普通, 多くの人々は FORTRAN を勉強する時, 英語の勉強のように, 文法を厳しく教わりながら勉強するわけではないからである. どちらかというと, 日本語を覚えたときと同じ様に, 「見よう見まね」または 「試行錯誤」により FORTRAN を勉強するのが普通である. そうなると, 多くの人が生まれた土地の方言を, 方言と気づかずに覚えてしまうのと同じ様に, 特定の計算機の特定のコンパイラだけが解釈できるプログラムを平気で書いてしまうことになる.

FORTRAN コンパイラは通常, 標準語としての FORTRAN77 規格 $+\alpha$ の機能を持っている. この$+\alpha$ の部分は, 拡張機能と呼ばれ, このような拡張機能を持つコンパイラは FORTRAN77 上位互換 (アッパーコンパチ) コンパイラと呼ばれる. これは, 標準語だけで書かれたプログラムでも, $+\alpha$ の機能を使ったプログラムも, どちらもコンパイルできるという意味ではコンパイラの 「機能が高い」のであるが, 逆に$+\alpha$ の機能を使ったプログラムは, そのコンパイラでないとコンパイルできないことになる. したがって, 我々は$+\alpha$の部分を 「方言」と呼ぶ.

日常会話の中では, 方言は特に問題にならず, むしろ標準語では表現できないニュアンスまで伝えることができて便利である場合が多い. これは, 方言が標準語の拡張機能として働いているためであるが, ビジネス上の会話で方言を使うと, とんでもない結果をもたらす. 多くの人は 「明日, 東京にいかずー」という約束をしたら, 明日, 東京にいくべきか否か迷うに違いない. (これは中部地方の方言で, 東京にいこう, という意味である)

これと同じ様に, FORTRAN の 「方言」は特定の計算機を使いこなすには便利な場合も多いが, むやみに使うと非常に危険である. しかし, 生まれた土地を離れたことがないと, 自分の言葉が方言かどうかわからないのと同じ様に, 特定の計算機しか使ったことがないと, 正確な FORTRAN の文法を身につけるのは難しい. なぜなら, いつも使っている FORTRAN77 のコンパイラがちゃんとコンパイルしてくれれば, そのプログラムは FORTRAN77 規格に準拠したプログラムであると思ってしまうからである. しかし, これは明らかな間違いである. どんなコンパイラでも大なり小なり 「方言」を持っており, その 「方言」を使っ

[8] 本節は地球流体電脳倶楽部ライブラリマニュアル 『MATH1」によるものである.

実行する　コンパイルして出来た実行型ファイルの名前を, 例えば mean としましょう.
　それをリダイレクションなしに実行すれば, キーボードから数値を読み込みディスプレイに結果を書き出します. この場合の実行例を挙げましょう.

```
geoph:/home2/s22516> mean
3.1
4
21.0
-0.3
    6.95000
geoph:/home2/s22516>
```

最初の 4 個の数値はキーボードからの入力で, 最後の行は結果です.
　入力する数値を先ほど作ったファイル *infile* から読み込む場合は

```
geoph:/home2/s22516> mean < infile
    6.95000
geoph:/home2/s22516>
```

のようになります. さらに, 結果をまだ存在していないファイル *outfile* に書き込む場合は

```
geoph:/home2/s22516> mean < infile > outfile
geoph:/home2/s22516>
```

として下さい. 結果を見たかったら, cat などで *outfile* の中身を表示させて下さい.

結果の利用　計算の結果をグラフにしたい時などは, 上記のような方法で数値の並んでいるテキストファイルを作り, それを gnuplot などに入力してやると良いでしょう. 絵を書くソフトについては, 6 章などを参照して下さい.

5.1.4　コンパイルとは

FORTRAN は人間が計算機に何かをさせるために指示を与える言語ですが, FORTRAN そのものを計算機は直接理解することができません. 計算機が直接理解できる言語は “機械語” だけです. そのため, FORTRAN を機械語に翻訳してやる必要があります. これを行なうのが FORTRAN コンパイラーです.
　5.1.2 節での “コンパイル” という言葉は, 正確な意味でのコンパイルと, “リンク” と呼ばれる操作を含んでいました. f77 はコンパイルとリンクを順次行なうプログラムです. リンクとは何かは, ここでは解説しません.
　コンパイラーに与えるプログラムのことを (FORTRAN の) “ソース · プログラム”, コンパイル · リンクの結果が書き込まれているファイルを “実行型ファイル” と呼びます. 実行型ファイルは, (シェルの) プロンプトに対してその名前を入力するだけで実行されます.

5.2　FORTRAN77 規格について

FORTRAN は 1950 年代から使われている, もっとも古い高級言語である. 1966 年に FORTRAN66 として規格が定められ, その後, 1978 年に全面的に改訂されて FORTRAN77 となった.
　このように, 長い歴史を持つ言語だけに, FORTRAN で書かれたライブラリーは膨大な量にのぼる. 反面, C 言語など新しい言語が数多く存在する現在では, 古い歴史を引きずった堅い (融通の効かない) 言語と

プログラムを書く　やはりエディターを使って,

```
      program  mean
c
      integer     ndat
      parameter  (ndat = 4)
c
      real        dat, sum
      integer     i
c
      sum = 0.0
      do 100  i = 1, ndat
        read(*,*)  dat
        sum = sum + dat
  100 continue
      write(*,*)  sum / real(ndat)
c
      stop
      end
```

のようなプログラムを書きます.

標準入力から値を読んでいるのは `read(*,*) dat` です. `read` 文の一般的な書き方は

`read`(*from where to read*, *format*) *variable*

ですが, *from where to read* の部分に * を指定すると読むのは標準入力からになります. *format* (書式[6]) の所の * は, 書式は 「適当に」しろ, という指定です. また, `read` 文 1 個が入力の 1 行に対応します. 結局, `read(*,*) dat` は, 標準入力から入ってくる文字の並びの 1 行を実数 (`real`) を表すものと解釈し, その値を実数型変数 `dat` に格納する, という働きをします.

標準出力に結果を書き出しているのは `write(*,*)` *variable* です. 書き方は `read` 文と同様です. つまり, この `write` 文の働きは, *variable* の値を (「適当な」書式の[7]) 文字列で表しそれを標準出力に書き出す, ということです.

プログラムの最初から順に簡単に説明します.

.....

integer ndat　`ndat` は整数型であるという宣言.

parameter (ndat = 4)　`ndat` を 4 という値の定数にせよ, という宣言. (ここでは, 入力する数値の個数が 4 だから.)

real dat, sum　`dat` と `sum` は実数型の変数であるという宣言.

.....

do 100 i=1,ndat ··· **100 continue**　(`i` の値を 1 から `ndat` まで変えながら) ··· の部分を 4 回繰り返す.

··· **sum / real(ndat)**　`ndat` は整数型なので, 実数型変数 `sum` を割る前に実数型に変換しておく.

[6] FORTRAN の文法書を参照せよ.
[7] 例えば, 何桁目を四捨五入するのか, などを FORTRAN のシステムに任してしまうということ

```
13 7b
```

という 16 進数の並びになる.

の 2 通りがあります. 前者のやり方でデータが格納されているファイルを “テキストファイル (text file)”, 後者の方を “バイナリーファイル (binary file)” といいます. ここで注意して欲しいのは, **UNIX** はテキストファイルとバイナリーファイルを区別しないということです. どちらも, UNIX にとってみれば単なる 16 進数の並びです[5]. この区別はそのファイルを使う側の区別でなのです.

エディターで作られたファイルはテキストファイルです. 例えば,

```
4987 UNIX
```

という 9 個の文字 (空白 1 個も含む) をエディターを使ってファイルにセーブしたとすると, そのファイルの中身は,

```
34 39 38 37 20 55 4e 49 58
```

という 16 進数の並びになります (最後の 10 桁は, 空白, 'U', 'N', 'I', 'X' のコード).

テキストファイルは内容を画面に簡単に表示することができます. 例えばコマンド cat を使って,

```
geoph:/home2/s22516> cat file
```

とすれば, *file* の内容を表示することができます.

> cat はファイルの中身をそのまま標準出力に送るコマンドです. 今の場合 (リダイレクションしてないので) 標準出力はディスプレイに接続されています. そして, ディスプレイというのは送られてくる 16 進数の並びを文字のコードだと解釈して, そのコードに対応する文字を画面に描く所だと思えば良いのです.

バイナリーファイルを編集したりその内容を表示したりすることは一般に簡単ではありません.

テキストファイルの欠点は, バイナリーファイルに比べて大きくなることです. そのため, 膨大な量の数値を扱う場合にはバイナリーファイルを使わざるを得ません. しかし, 取り扱いが楽であることやシステムに依存しにくいことから, UNIX 上ではデータはテキストファイルで保存し受渡しするのが基本です.

標準入出力を使ったプログラム

ここでは, テキストファイルからデータを読み込み, 結果をテキストファイルに書き込むプログラムの書き方を解説します. その際にリダイレクションを使い, プログラムの中でファイル名を指定することはしません. バイナリーファイルの使い方や, プログラム中でファイルを指定する方法については解説しません.

例として, テキストファイル *infile* に並んでいる数値を読み込んでその平均値を計算し, それを *outfile* に書き出す, というプログラムを取り上げます.

数値のファイルを作る　エディターを使って,

```
3.1
4
21.0
-0.3
```

のように, 1 行に 1 個ずつ数値を書き, ファイル *infile* にセーブします. ここでは, 数値は 4 個です.

[5] これらを区別する UNIX もあるそうですが, 少なくとも Sun の UNIX は区別しません.

標準入力 · 標準出力 — リダイレクション

UNIX の標準入出力という概念とその有用性について説明します.

あるコマンド[4] *command* を実行するには, 通常,

```
geoph:/home2/s22516> command
```

のように, コマンド名をシェルのプロンプトに対して入力します. このようにして実行されたコマンドに対しては, シェルは 「標準入力」と 「標準出力」を割り当てます. そして (上のようにシェルに対して何も指示をしない時には), 「標準入力」はキーボードに, 「標準出力」はディスプレイに接続されています.

ここで, *command* は, 標準入力から何かを読み込み, 標準出力に何かを書き出すように作られているコマンドだとしましょう. すると, 上のように実行した時には *command* は, キーボードから読み込みディスプレイに書き出すことになります.

しかし, シェルに対して入出力をファイルに切替えるよう指示をすることができます. このような切替のことを “リダイレクション (redirection)” といいます. 標準入力を *infile* に接続させるためには,

```
geoph:/home2/s22516> command < infile
```

とします. こうすると, 書き出すのはディスプレイのままですが, 読み込むのは *infile* からになります.

また, 標準出力を *outfile* に接続させるためには,

```
geoph:/home2/s22516> command > outfile
```

とします. *outfile* が存在していないファイルの場合は新たに作られますが, 既に存在している場合は上書きされます. 既に存在しているファイルの後ろに続けて書き込みたい場合には,

```
geoph:/home2/s22516> command >> outfile
```

とします.

標準入力を *infile* に, 標準出力を *outfile* に接続させるには,

```
geoph:/home2/s22516> command < infile > outfile
```

などとします.

以上のようなリダイレクションの仕掛けによって, (入力が 1 つ, 出力が 1 つの場合には) プログラム — それをコンパイルした結果が *command* になる — の中でファイル名を指定する必要はなく, いつも標準入出力を使うようにしておけば良いということになります.

テキストファイルとバイナリーファイル

あるファイルから数値を読み込み, プログラムで処理したいとしましょう. この時, そのファイルに数値を格納しておくのに大きく 2 通りのやり方があります. 例えば, 4987 という整数値を格納する場合,

1. ’4’ ’9’ ’8’ ’7’ という 4 個の文字の並びとして格納する. 各文字には 16 進数で 2 桁のコードが割り当てられていて, この 4 文字のコードは順に 34, 39, 38, 37 なので, ファイルの中身は

   ```
   34 39 38 37
   ```

   という 16 進数の並びになる.

2. その整数そのものを格納する. 4987 は 16 進数で 137b なので, ファイルの中身は,

[4] 実行型ファイルのことだと思って下さい.

```
program  sample
write(*,*)  'hello, world'
stop
end
```

これは, 画面に hello, world という文字を書くプログラムです.

> program sample はこのプログラムに *sample* という名前をつけろという宣言です. 名前は 6 文字以内の好きな名前で構いませんし, この宣言はなくても構いません. write(*,*) は画面にいろいろなものを書き出す命令です. この 2 個の * の代わりに, 書き出す場所や書式の指定をするものを置くことができます. stop はプログラムの実行を止めて OS に戻るという指示です. stop はプログラム中のどこにでも置くことができます. end はプログラム単位 (ここではメインプログラムという単位. 他に, サブルーチンや関数という単位がある.) の終わりであるということを表します.

そして, このプログラムを書き込んだファイル名を example.f とします.

5.1.2　コンパイルし実行する

コンパイルとは何かということは 5.1.4 節に譲ることにして, ここではそのやり方だけを説明します.

前節の例 example.f をコンパイルするためには, シェルのプロンプト (ここでは, geoph:/home2/s22516>) に対して

```
geoph:/home2/s22516> f77 -o example example.f
```

と入力して下さい. プログラムに間違いがなければ, 実行型ファイル[3] example ができます.

これを実行するためには, プロンプトに対して

```
geoph:/home2/s22516> example
```

と入力します. すると,

```
geoph:/home2/s22516> example
hello, world
geoph:/home2/s22516>
```

となるはずです.

5.1.3　最も簡単な入出力

実際のプログラムに最も良くあるのは,

外から何かデータを読み込み,
それを元に何か計算をし,
結果を外に出す (見せる).

というタイプのものです. 入出力ができなくては何もできないに等しく, しかも, 問題の多い部分なので, この節では, 最も簡単な入出力の仕方を解説します.

しかし, その前に, — FORTRAN に関する説明ではなく UNIX に関する説明になりますが —

リダイレクション
テキストファイルとバイナリーファイル

について解説しておきます.

[3] やはり, 5.1.4 節を参照して下さい.

第 5 章

プログラム言語

A program runs on which it programmed, Not on programmer's will.

language lawyer n. A person, usually an experienced or senior software engineer, who is intimately familiar with many or most of the numerous restrictions and features (both useful and esoteric) applicable to one or more computer languages. A language lawyer is distinguished by the ability to show you the five sentences scattered through a 200-plus-page manual that together imply the answer to your question "if only you had thought to look there".

the new HACKER'S DICTIONARY, MIT PRESS

5.1　最低限の FORTRAN 入門

この節では, まず, FORTRAN を全く使ったことのない人を対象に,

プログラムを書き,
コンパイルし,
実行する

ということができるようになるための最低限の解説をします.

その次に, 簡単な入出力を伴うプログラムの書き方を解説します.

前提とすることは, 何でも良いからエディター (emacs など) を使い文書を編集しファイルに書き込むことができること, です.

FORTRAN の文法や仕様については, 数多くの解説書が出ていますので (例えば, 浦, 1982) ここでは全く解説しません. ただし, "FORTRAN 77"[1] の解説書を選ぶようにして下さい.

5.1.1　プログラムを書く

emacs などのエディターを使って FORTRAN のプログラムを書き, ファイルにセーブします. ファイル名は `.f` で終わっていなくてはなりません.

ここでは, 次のようなプログラムを作ったとしましょう[2].

[1] FORTRAN 66 は古い規格. JIS FORTRAN は実質的には FORTRAN 77 と同じ.
[2] 各行の先頭には 6 個の空白をが必要です.

```
Message from Talk_Daemon@geoph at 03:42 ...
talk: connection requested by yoshida@geoph.geoph.s.u-tokyo.ac.jp
talk: respond with:  talk yoshida@geoph.geoph.s.u-tokyo.ac.jp
```

のようなメッセージが表示されます. これを見れば, 誰がどこの端末から talk をかけてきたかがわかる (今は yoshida が geoph という計算機から talk をかけた) ので, talk がかかってきた計算機と同じ計算機の空いているウィンドウで,

```
talk: respond with:  talk yoshida@geoph.geoph.s.u-tokyo.ac.jp
```

にある通り

```
talk yoshida@geoph.geoph.s.u-tokyo.ac.jp
```

とやってあげれば, コネクトが完了し, talk が使える状態になります. kterm のマウスによる cut & paste を利用すると良いでしょう. 後は, ローマ字でも英語でも, 好きなように書いて遊んで下さい.

終る時は, Ctrl-c です.

write

write の使い方も talk とほとんど同じです. 手順は, まず相手を rusers などで探す, 相手のいる端末に rlogin する, メッセージを送る, となります. write は talk と違い, 異なった計算機間で情報をやりとりすることはできません. ですから相手がいる計算機に rlogin しなければなりません.

```
write s22516
```

として, メッセージを書き, リターンキーを叩くと, 相手にメッセージが送られます. write をかけられた側も, そのままメッセージを書いてリターンキーを叩けば, ちゃんと相手にメッセージをおくることができます.

mesg

時と場合によっては write や talk をされたくない時もあります. そういう時は

```
mesg n
```

としておけば, 相手が talk 等をかけようとしてもつながりません. もちろん, 自分の端末には何のメッセージもでませんので, 自分の仕事が邪魔される事はありません.

では, s22516@vsgold に talk をかけられるかを調べる方法ですが,

```
finger @vsgold
```

とすればいいだけです. TTY の所に * マークがついていたら, その人は talk 等を受け付けないと言う事を示します.

```
[vsgold]
Login       Name              TTY Idle    When    Where
s22516   Tibutu Tarou         *p0      Tue 15:35  vsiron:0.0
```

この場合だと s22516@vsgold は talk 等を受け付けていません.

```
Office: S3-213, 4291            Home phone:
Directory: /home/users1/ohtake          Shell: /usr/local/bin/tcsh
Last login Thu Apr  1 02:49 on ttyp0 from wapiko
No unread mail
Plan:
---------------------------------------
Dept. of Earth and Planetary Physics, Univ. of Tokyo, Japan.
Address : 2-11-16 Yayoi, Bunkyo-ku, Tokyo, 113 JAPAN
Office  : Rm. 213, 3rd bld. of Faculty of Sci., Asano district.
Tel. No.: +81-3-3812-2111 ext.4291.
e-mail  : ohtake@geoph.s.u-tokyo.ac.jp
```

のように, Plan:の後に自己紹介を挿入してくれます.

4.4.4 talk & write

talk と write は, 離れたホストに座っている人間の間で, 簡単な会話が出来るようにした機能です. talk は, 一つのウィンドウを上下二つに割って, 上が自分, 下が相手のメッセージを表示するようにしたものです. 一方 write は, こちらのメッセージを相手に送りつけるものです.

マナー

talk も write も, こちら側からまず, 「これから talk をしたいんだけど. 」という意思表示を, 相手のホストに送らねばなりません. この時, 相手はどんな作業をしているか, こちらにはわからないわけですから, もし相手が忙しい時には遠慮した方が better でしょう.

中には, 「あちこちから talk がかかってきて仕事が何にも出来ない !」と嘆いている人もいるくらいですからね.

talk

まず, 話したい相手がどこにいるか調べなければいけません. これには, 先ほどの finger の他に, rusers や rwho といった命令があります. どんな命令かはやってみればわかるでしょう.

talk は次のようにして起動します.

```
talk 相手のユーザ名@相手の login している端末名
```

さて, 今話をしたい相手 (s22516 と仮定しましょう) が, vsgold.geoph.s.u-tokyo.ac.jp という端末にいることがわかったとします.

```
talk s22516@vsgold.geoph.s.u-tokyo.ac.jp
```

とします.

これで, 準備は完了です. 画面が上下二つに分かれて, 後は相手が反応するのを待つだけです.

これまでは, talk をかける側の話でした. では, talk がかかってきた時はどうしたらよいのでしょか. talk がかかってくると, 「ピッ」という発信音がしてどこかのウインドウに

```
On since Apr  1 15:10:57 on ttyp3 from earth
Mail last read Thu Apr  1 15:11:02 1993
No Plan.
```

のように表示されます. 意味はわざわざ説明するまでもないでしょう.

次に, 計算機を指定するやり方を説明しましょう. これは,

```
finger @計算機名
```

とします. 指定した計算機に現在誰がログインしているか, 或は最近誰がログインしたかを調べるためのものです. 例えば,

```
finger @tansei
```

と打てば,

```
[tansei]
  User     Real Name         What    Idle  TTY  Host      Console Location
aono-pro Aono-pro                    0:05   pe tansei1
b39878   B39878                             pa tansei1
huematsu Huematsu                           p0 tansei1
kurosawa Kurosawa                           p7 tansei1
m-kimura Michio KIMURA                      p4 tansei1   Tsukuba Uni 0298-
manager         ???                  0:19   p3 tansei1
mshino   Masanao Shinohara           0:49   p2 tansei1   ORI.MarSeis 3376-
nakanisi 'Higepy' Hideya             0:02   p5 tansei1   Inoue Lab.N ext.6
nakayama Nakayama                           p9 tansei1
tnchen   Tnchen                      0:01   pc tansei1
toda     Shinichi Toda                      p1 tansei1
tomohiro Tomohiro                           p8 tansei1
y-taka   Y-taka                             p6 tansei1
```

のように表示されて, tansei というマシンに現在誰がログインしているかがわかります.

また,

```
finger @geoph
[geoph]
No one logged on
```

というような表示もあります.

上の例の Name のフィールド, または `In real life:` の後に書かれているものは歴史的経緯から GECOS フィールドと言います. ここに本人の名前などを書きます. コマンド `chfn` というコマンドで英文字を入れて下さい. やり方は 2.2.10 節に書いてあります. 必ず登録しておいて下さい.

.plan というファイルに簡単な自己紹介を書いておくと,

```
% finger ohtake@geoph

[geoph]
Login name: ohtake                      In real life: Ohtake Kazuo
```

ただし, MS-DOS は Shift-JIS , SUN は JIS, tecc は EUC が標準ですので, 日本語を含むファイルの転送には, nkf などで対処する必要があります.

ftp を実行するとき, ftp マシン名 では, うまくいかないことがあります. このようなときは, ftp IP address とする必要があります. 例えば, 研究室の PC-98 等を使って geoph にファイルを転送するときは, ftp 133.11.24.1 とします.

4.4.2 anonymous ftp

anonymous ftp site

anonymous ftp サイトとは, アカウントを持つ人以外の ftp アクセスを受け付けるマシンのことを指します. ここには, 多くの有益なファイルなどが蓄積されています. 私達は, この恩恵にあずかることができます. 例えば, ftp.u-tokyo.ac.jp は anonymous ftp を受け付けています.

では, 実行例を示します.

```
ftp ftp.u-tokyo.ac.jp
```

もしくは

```
ftp 130.69.254.254
```

とします.

name を尋ねられたら, anonymous と答えます. password には, もし, あなたが, s22516 で geoph からアクセスしたならば, s22516@geoph.s.u-tokyo.ac.jp などと自分の有効な E-mail アドレス, またはアクセスしたマシンの正式名称を答えれば良いでしょう. 実は, anonymous ftp の場合, password は関係ないのですが, 管理者はどのような人がアクセスしているのか記録を取るのでこう答えるわけです. また, 幾つかの ftp site では password で入力されたマシン名の問い合わせをして, それが存在した時のみにアクセスを許す所もあります. 決して, geoph の本当の password を入力してはいけません.

anonymous とは, 「匿名」という意味ですので, get できるファイルもそれだけ制限を受けますが, ftp.u-tokyo.ac.jp や utsun.s.u-tokyo.ac.jp には結構いろいろなファイルがそろっているので便利です.

4.4.3 finger

finger

finger は, 他のユーザに関する情報 (ログインネーム, 名前, 最後にログインした日付, メールをちゃんと読んでいるかどうか, 等々) を手に入れるためのコマンドです. つまりもちろん, 自分に関する情報も手に入れらます.

具体的な使い方を示します. 一番簡単な使い方は,

```
finger ユーザ名
```

とします. 例えば,

```
finger yoshida
```

とします. そうすると,

```
Login name: yoshida                    In real life: Yoshida Shigeo
Directory: /home/yoshida               Shell: /usr/local/bin/tcsh
```

4.4 Network 関連命令

4.4.1 ftp

ftp は 対話形式でリモートホストとの間で, ファイルを転送するプログラムです. もっと詳しくいえば, ファイルを転送する信号規格を ftp(File Transfer Protocol) といい, それを実行するプログラムも ftp と言っています.

起動

```
ftp ホスト名
```

とします. 例えば, tecc から,

```
ftp geoph
```

とする場合, geoph が存在して立ち上がっていれば, 相手ホスト (この場合 geoph) における ID, パスワードを聞かれます. 無事入れれば, ftp> というプロンプトが出ます.

終了

終了の命令は quit です.

転送に使う命令

使える命令は ?と言ってやれば, 一覧を見ることができます. 主に

get 相手ホストからファイルを持ってくる.
put 相手ホストにファイルを持っていく.

でやりとりする[5]わけですが, その際に必要な ls, cd, etc.. の命令はもちろんあります. ただ, 接続を要求したホストでのこのような命令の実行には 前に! をつけます. 場合によっては, l かもしれません.

binary ファイルの転送には

```
ftp>binary
```

として, binary モードで, 実行しましょう.

便利な mput, mget なんていう一括転送コマンドもあります. 例えば, *.c というファイルをすべて送りたければ,

```
ftp>mput *.c
```

とします.

MS-DOS との交信

教育用計算機センターの FM-R を使って, MS-DOS で作ったファイルを tecc や secc[6] に送ることもできます. 但し, このときは, fnsstart とする必要があるでしょう. このことは, 教育用計算機センターの利用の手引きに書いてあります.

[5] これでは, 1 つずつしかファイルをやりとりできません.
[6] 教育用計算機センターの S ワークステーションのこと.

C-c C-s a 記事の投稿者名でソートする
C-c C-s C-a 同じ
C-c C-s s サブジェクトをアルファベット順にソートする
C-c C-s C-s 同じ
C-c C-s d 記事の投稿日付でソートする
C-c C-s C-d 同じ
= *Subject* を全画面にする
G このグループの全サブジェクトを表示する
w ^L などでのページ止めをしない
C-c C-r 暗号 Rot 13/47 をかける
g この記事を表示する
t 表示するヘッダの形式を切り替える
v 記事のすべてのヘッダを表示する
C-d RMAIL モードでダィジェストを見る
a 記事をポストする
f フォローアップ記事をポストする
C 記事をキャンセルする
r メールでリプライする
m 別ウィンドウでメールを出す
o 記事を保存する (形式は, Unix ファイル, mbox, RMAIL,MH-folder
などがある. デフォルトは Unix ファイル)
C-o mbox 形式で保存する
| 記事の内容をパイプする
ESC k ローカル KILL ファイルを編集する
V GNUS のバージョンを表示する
q *Subject* を抜ける (記事の未読状態を記録)
Q 何もしないで *Subject* を抜ける
? シンプルなガイダンスを表示
C-h m *Help* を表示
C-c TAB *Subject* バッファの Info を表示 (オンラインマニュアル)

*** KILL ファイルの編集* – Edit KILL file バッファ**

C-c C-c KILL ファイルを保存して, 元のバッファに戻る
C-c C-i Edit KILL file モードの Info を表示する

- KILL ファイルには, グローバル KILL ファイルとローカル KILL ファイルの 2 種類があります. これらのファイルには, Emacs-Lisp が記述できます.
- グローバル KILL ファイル名は, 通常 ~/News/KILL です.
- ローカル KILL ファイル名は, gnus-use-long-file-name によってファイル名が変わります.
  gnus-use-long-file-name が none-nil の場合 (デフォルト) には,
  通常 ~/News/news.group.name.KILL などです.
  gnus-use-long-file-name が nil の場合には, 通常~/News/news/group/name/KILL になります.

***記事選択*– *Subject* バッファ**

SPC	記事を読む (*Article* 表示後は，次ページを表示)
DEL	前ページを表示する
RET	一行だけスクロールアップ
n	次の未読記事を表示
p	前の未読記事を表示
N	次の記事を表示
P	前の記事を表示
ESC C-n	同じサブジェクトの次の記事を表示
ESC C-p	同じサブジェクトの前の記事を表示
C-c C-n	ダイジェスト記事で，次のダイジェストに移動
C-c C-p	ダイジェスト記事で，前のダイジェストに移動
C-n	次のサブジェクトに移動
C-p	前のサブジェクトに移動
ESC n	次の未読サブジェクトに移動
ESC p	前の未読サブジェクトに移動
.	このグループの先頭の未読記事に移動
/	incremental search
s	記事の内容を incremental search する
ESC s	記事の内容を後へ regexp search する
ESC S	記事の内容を前へ regexp search する
<	記事の先頭に移動
>	記事の最後に移動
j	指定の記事番号に移動
J	指定の記事番号の内容を表示
^	この記事の親記事に移動
ESC r	指定された記事 ID を持つ記事を表示
l	最後に読んだ記事に移動
u	未読マークを付けて，次に移動
U	未読マークを付けて，前に移動
d	読んだことにして，次に移動
D	読んだことにして，前に移動
ESC u	未読マークの undo して，次に移動
ESC U	未読マークの undo して，前に移動
k	同じサブジェクトの記事をすべて KILL して，次の未読記事に移動
C-k	同じサブジェクトの記事をすべて KILL する
c	すべての記事を Catch Up して，グループを抜ける
C-t	*Subject* バッファの折返しモードを変える
x	未読記事のサブジェクトだけを表示する
X	指定されたマークが付いている記事を消す
&	指定されたヘッダの内容を持つ記事に対してコマンドを実行する
C-c C-s n	記事番号でソートする
C-c C-s C-n	同じ

***グループ選択*– *Newsgroup* バッファ**

SPC	グループを選択する
=	サブジェクト一覧を表示する
j	指定グループに移動する (グループ名は，minibuffer で入力)
n	未読記事のある次のグループに移動
p	未読記事のある前のグループに移動
DEL	同じ
N	次のグループに移動
C-n	同じ
P	前のグループに移動
C-p	同じ
/	incremental-search
<	バッファの先頭に移動
>	バッファの最終に移動
u	現在のグループを未購読 (unsub) にする / しない (トグル)
U	指定のグループを未購読 (unsub) にする / しない (トグル)
c	全部読んだことにする (Catch Up)
l	グループ名の再表示
L	すべてのグループ名を表示 (unsub も含む)
g	新たに増えたニュースをチェックする
R	.newsrc の再読込みをする
b	Check bogus newsgroup
r	point と mark の間のグループを Narrowing する (C-x w で元の画面になる)
a	記事をポストする
ESC K	グローバル KILL ファイルを編集する (C-c C-c で元の画面になる)
ESC k	ローカル KILL ファイルを編集する (C-c C-c で元の画面になる)
V	GNUS の版数を表示する
x	.newsrc を保存する
s	同じ
z	GNUS 起動前のバッファに戻る (仕事に戻る :-)
q	GNUS を終了する .newsrc を保存することを確認する
Q	GNUS を終了する .newsrc を保存しないことを確認する
?	シンプルなガイダンスを表示
C-h m	*Help* を表示
C-c C-i	*Newsgroup* バッファの Info を表示 (オンラインマニュアル)

- デフォルトのサーバホスト名 gnus-nntp-server が指定されていないならば minibuffer に問い合わせます. また, C-u などでアーギュメントを指定して GNUS を起動すると同様に問い合わせます.

- ~/.newsrc-SERVER ファイルが存在していれば, ~/.newsrc ファイルの代わりにそれを使用します. SERVER は, NNTP サーバのホスト名です.

- 記事をポストする時に指定した DISTRIBUTION に合わせて, ~/.signature-DISTRIBUTION ファイルが存在していれば, ~/.signature の代わりに使われます.

とすることでもできます. この場合も, ニュースから抜けておく必要があります. 以下, C-<key> は, Contorol キーを押しながら, <key> というキーを押すことを意味することにします.

目次が現れたら, 読みたいニュースグループを決め, 読みたくないニュースグループには, u で “読まない” の印をつけます. やっぱり, 読みたかったという時は, もう 1 度 u とすれば, “読まない” の印が消えます. “読まない” の印をつけたニュースグループは次からは, 目次にでませんが, 出したくなったら, L とすれば, 全ニュースグループが表示されます. ニュースグループの隣に出ている数字は, 読んでいない記事の数です. これを 0 にするには, c y とすれば O.K. です.

地球惑星物理学科のニュースは先頭に geoph とついたニュースグループです. たくさんニュースを読むように設定していると, このニュースグループを探すことも結構手間取ります. emacs の C-s を使って geoph.* というニュースグループを探すことがあります.

ニュースグループを選ぶ時はスペースキーを押します. すると, 上に, 要約が出て, 下に記事が出ます. 場合によっては, 新しい記事から何番目の記事まで読むか聞いてくるので適当に答えます.

ただ順番に読んでいきたいのであれば, スペースキーを押せば, 次の内容に移ります. カーソルキーは, 要約の書かれているバッファーにあると思いますが, そこで操作すればよく, 記事のバッファーに移る必要はありません.

記事を保存するときは, o または, C-o とします.

記事を読まない時は, d, 前の記事に戻る時は, p, あるいは, C-p , 次の記事にいくときは, n, または, C-n として, スペースキーで記事を選択します. 詳しくは, 4.3.6 節を参照してください.

記事にフォローするときは, f, 記事を投稿する時は, a, とします. 筆者にメールを送る時は, r です. 記事を投稿する時には, 本当に記事を投稿するか問い合わせてくるので, 指示通りに答えます. 記事にフォローしたり, 返事を書くときフォローする記事を挿入する場合は, C-c C-y とします. この辺りは, RMAIL に似ています. 実際に記事を書く時も, 電子メールを書く時と同じように書けばよいのです.

最後に記事をポストする時は, C-c C-c とします. このとき, ホームディレクトリーに, .signature があると, ポストする記事の末尾にその内容が自動的に加えられます. 自動的 に加えられるので, emacs の電子メールシステムのように, わざわざ C-c C-w などとして, .signature を記事の最後につける必要はありません.

4.3.5　アクセスできるニュースサーバー

geoph からアクセスできるニュースサーバーを紹介します (これが全てではないです.).

```
climate3.ccsr.u-tokyo.ac.jp
news.u-tokyo.ac.jp 又は ns.nc.u-tokyo.ac.jp
tecc.ecc.u-tokyo.ac.jp
```

4.3.6　簡単なマニュアル

ここで, tecc の bbs の利用者提供の手引きに載っていたマニュアル[4] を掲載します. 以下で ESC は, M と読み替えても構わないでしょう.

[4] GNUS 3.11 キーバインド ・リファレンス
First made 88.11.18 saka@pf.fujitsu
Revised 1.1 saka@pf.fujitsu
Revised 1.2 shige@csk.JUNET
Revised 1.3 saka@pf.fujitsu.junet
for GNUS 3.11

他の計算機のニュースを読むこともできます. その計算機にアカウントを持っていなくても構いません. ただ, アカウントが無い計算機の上にあるニュースを読むというのは他人の計算機の資源を無断に使用させてもらっているのですから, それなりの仁義を心得て下さい. 一般にはアカウントが無いマシンのニュースを読むのは正式には許可は出していなく, 好意的に管理者の人がガードをかけていないところが多いのです.

他の計算機を読みにいく場合の手順は次の通りです. まず自分のホームディレクトリに .emacs が存在することを確認してください.

```
geoph:/home2/s22516> ls -a

./              .cshrc          .mh_profile     .xinitrc        .xtenkeyon
../             .emacs          .rhosts         .xkeymap.japan
.Xdefaults      .login          .tcshrc         .xsession*
.Xresources@    .logout         .twmrc          .xtenkeyoff
```

この .emacs は emacs の個人用設定ファイルです. emacs を使って .emacs の最後に以下を記入してください (編集画面に .emacs を読み込むには 2.9.6節を参照, 編集後ファイルをセーブするには 2.9.5を参照) .

```
(defun gnus-u-tokyo ()
  "access NNTP server of Univ of Tokyo by gnus"
  (interactive)
  (setq gnus-news-system 'Cnews)
  (setq gnus-nntp-server "news.u-tokyo.ac.jp")
  (setq gnus-nntp-service "nntp")
  (setq gnus-startup-file "~/.newsrc-u-tokyo")
  (gnus))
```

完了したら emacs を終了し, 再度立ち上げなおしてください. そして emacs で

```
M-x gnus-u-tokyo
```

とおまじないを唱えると, めでたく news.u-tokyo.ac.jp のニュースを読むことができるようになります. 以後は .emacs の編集をしなくても最後のおまじないだけで読むことができます. 他の計算機のニュースを読む場合にも同様のことが必要です.

さてニュースに入るとまず目次 (Newsgroup buffer) が現れます. もし, このときニュースグループが整列されていないと見にくいので, 一旦 q として, ニュースから抜けましょう. ニュースから抜ける際に本当に抜けてよいかどうか尋ねてきますので, このときは y と答えます. このとき shell で, 例えば,

```
sort .newsrc-u-tokyo > foo
mv foo .newsrc-u-tokyo
```

とすればニュースグループの整列ができるはずです. ここで, また, M-x gnus としま す.

この操作は, emacs の中でも出来て,

C-x C-f	
.newsrc-u-tokyo	ファイル名入力
M->	全体にリージョンを設定
M-x sort-line	リージョン内のソート
C-x C-s	ファイルのセーブ

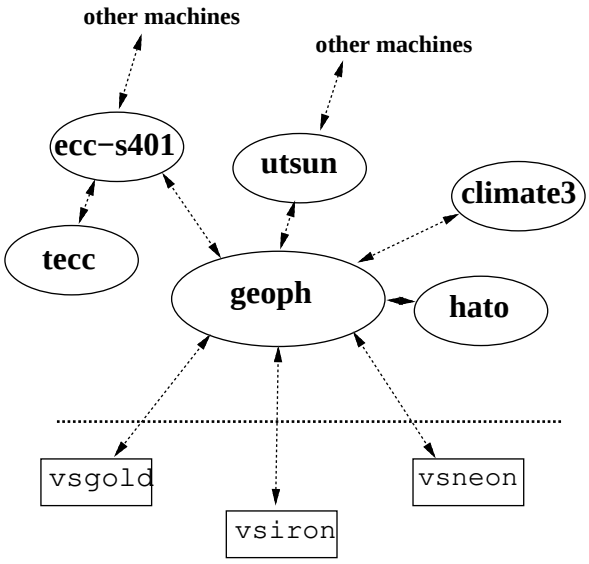


図 4.1: Network News System

- Subject には漢字を使わないのが通例です. MIME [1] を利用する方法もありますが, 使わないことにこしたことはありません.
- 署名は 4 行以内にすることをお奨めします. これは強制ではありませんが, このことで幾つもの論争が起きていることを知らせておきましょう.
- from 行には有効なメールアドレスを書いて下さい. 普通は自動的に付記されます.

4.3.3　教育用計算機センターのニュース

tecc で始まるニュースグループ名を持つニュースは教育用計算機センターが管理しているニュースです. 学科, サークル, 計算機, 趣味などのグループがあります. 昔は tecc という計算機だけしか読めなかったのですが, 現在は学内の多くの計算機が記事を交換するようになっています.

4.3.4　ニュースを読むには

ニュースを読む方法としては, rn を使うなど色々な方法があるようですが, ここでは emacs の中から GNUS というプログラムを用いてニュースを読む方法を紹介します.

まず, X-Window 上で[2] emacs を開きます. emacs の中で,

```
M-x gnus
```
[3]

とすると, ニュースを読むことができます.

ニュースを読む時, 前読んだ記事を何度も読むのは無駄ですので, どれだけニュースを読んだかを記憶しておき, これをもとに新たなニュースだけ読むようになっています. 何もしないと記憶場所は `.newsrc` になります.

[1] Multipurpose Internet Mail Extensions の略
[2] X-Window に限らず, 漢字の読める emacs が立ち上がればそこでニュースを読むことができます.
[3] `M-x` とは, メタキーを押しながら x を押すか, ESC キーを押してから x を押すことを意味します.

ここで注意ですが, 時々人の環境をコピーして, .forward をも持っていってしまうひとがあります. そうするとコピーされた人の所へコピーした人の手紙が送られてしまいます. 知っている同士でも気まずい手紙というのはあるものですが, 知らない人の手紙が来た時にはニュースで曝し物にする以外どうしようもありません. (電子メールで知らせるわけにもいかないのです. お分かりですね.) 充分気をつけて下さい.

4.2.7 メールアドレスの別名について

メールのアドレスはたいてい長く間違い易かったり, 毎回打つのは大変です. 別名を登録しておくと便利です.

mh の場合, 別名を登録するファイルの名前は任意ですが, ~/.mh_profile にそのファイル名を書いておかなくてはなりません. 例えば,

```
Aliasfile: /home/ichiro/.aliases
```

などと書きます.

例えば, yoshida@geoph.s.u-tokyo.ac.jp を magnetic という名前の別名登録をする場合は, ~/.mh_profile に書いたファイル (/home/ichiro/.aliases) に,

```
magnetic; yoshida@geoph.s.u-tokyo.ac.jp
```

と書いておきます. こうしておくと, M-x mh-smail をたちあげた時に, To: に magnetic と書けば, yoshida@geoph.s.u-tokyo.ac.jp に送ることができます. ただし, 同じマシンに magnetic という人がいるときには注意が必要です.

mail コマンドや RMAIL では別名の登録のしかたが異なるので注意して下さい.

4.3 Network News

4.3.1 Network News とは

Network News というのはいわゆる電子掲示板 (BBS) のようなものです.

Network News の特徴は, 電子掲示板のように一つの計算機に記事を置いておいて, 読者が皆そのマシンに login して記事を読むのではなく, 幾つかのマシンが記事を持っていて, その内のどれかを何らかの方法で読むというものです (図 4.1参照). そしてそれらのマシンが互いに記事を交換しているのです.

ですから, 記事があるマシンに投稿されても, すぐには他のマシンにあるとは限りません. また, その順番もマシンによって違います.

さらに, 全ての記事をやり取りする訳ではありません. 例えば page.* というニュースグループは日本のいくつかの大学の地球物理関係にしか流れていません. 一方, fj.* などはたくさんのマシンに流れていきます. 何を渡すかどうかは各マシンのニュースを管理している人達が決めます.

地球惑星物理学教室関係には, geoph というマシンで管理しているニュースグループ geoph.* と, 教育用計算機センターの地物のニュースグループ tecc.campus.sci.geophys があります. また, 日本のいくつかの大学の地球物理関係にながれているニュースグループに page.* があります.

4.3.2 学外のニュース

geoph から読める記事のうち, tecc.*, geoph.*, u-tokyo.* 以外は学外のニュースグループです. fj.* というのは主に日本で, その他は世界中で購読され, 投稿されています.

ここに投稿する時の注意をいくつか.

```
1  08/16 Alfred Hitchcock   Notorious <<Dear Ichiro; Thank you for
2  08/16 Ernst Lubitsch     To Be or Not To Be <<Hello. How are you.
3  08/30 Clint Eastwood     The Gauntlet <<Hi. Nice to hear from you
```

などと表示されます. 各行が一通のメールに相当し, 各欄は順に, 通し番号, 差出人の名前, 題目, メッセージの冒頭部分, です. 読みたいメールの行にカーソルを移動して, .(ピリオド) を入力すると, ウィンドウが分割されて下の部分にメールの内容が表示されます. n あるいは p で次のメールあるいは前のメールを表示させることができます. 保存しておく必要のないメールには d で削除マークを付けておきます. あとで x を入力すると削除されます. q で終了します.

返事を出す

返事を出すには, 「メールを読む」に示したようなメールの一覧のウィンドウで a を入力します. ミニバッファに `Reply to whom:` というプロンプトが出るので, 差出人に返事を出すときには `from` と入力します (他にも幾つか選択肢があるのですが, ここでは説明しません). そうすると, draft-?? バッファが開きます (`To:` と `Subject:` は自動的に付きます) ので, あとは 「メールを送る」に述べたのと同様に内容を書いて送ればよいです.

フォルダの活用

`M-x mh-rmail` を実行すると, 実は, メールはスプールと呼ばれる一時保存場所から `~/Mail/inbox` というディレクトリに移されます. 「メールを読む」に示したようなメールの一覧の番号は, 実はそれぞれのメールが保存されているファイルのファイル名でもあります (例えば Clint Eastwood からのメールは `~/Mail/inbox/3` というファイルに入っている).

inbox のようなディレクトリのことを 「フォルダ」と呼びます. フォルダは inbox 以外にもユーザーが自由に作ることができて, メールの内容や差出人などによって分類しておくことができます.

例えば, Clint Eastwood からのメールを private というフォルダに入れたいとします. そのためには, 当該メールの行にカーソルを移動して ^ を入力します. `Destination folder?` とプロンプトがでるので, private と入力します. private というフォルダが存在していない場合には, 作成するかどうか尋いてくるので, y と答えます. 最後に, x と入力すれば移動されます.

フォルダに入れたメールを読むためには, `M-f` と入力し, フォルダ名を入力します. 表示する範囲を尋いてきますが, とりあえずただリターンを入力すればいいでしょう. あとは, inbox を読むときと同じです.

4.2.6　メールの転送について

アカウントを複数のマシンに持っているような場合, あまり使わない方にメールが来ていると, メールを読むのが遅れる危険あります. このような場合には

```
.forward
```

というファイルを作っておくと便利です. このファイルの内容は, 例えば, このファイルを secc に置いて, geoph にメールを転送する場合, あなたの user 名が s22516 であれば,

```
s22516@geoph.s.u-tokyo.ac.jp
```

としておきます. これで, secc に来たメールは, geoph に送られることになります. ただし, .forward は他人に読めるようにしておかないと, 転送されないので, chmod +r .forward とする必要があります.

あまり大きなファイルを手紙で送らないようにしましょう.

電子メールは普通の郵政省の手紙のように文を送るだけでなく, いろいろな Unix のファイルをメールにして送ることができます. ですが, あまり大きなファイルを送ると相手の計算機やメールを中継してくれる計算機に不都合が生じることがあります. 大きくとも 50Kbyte くらいに収めましょう.

一行の長さには制限があることがある.

手紙を中継する計算機によっては一行の長さがある程度以上だと切れてしまうことがあります. 一行を漢字で 40 文字くらいを目安に改行をしましょう.

4.2.4 参考文献

これからは電子メールの使い方を説明します.
電子メールをやりとりするには, Unix 標準の mail, また emacs からの RMAIL , mh などを使います.
ほとんどの人が RMAIL や, mh を使っていることと思います.
ここでは mh を emacs (NEmacs あるいは Mule) のなかで使用する方法 (mh-e) を説明します. RMAIL については, Unix Magazine の 90 年 11 月号, もしくは GNU Emacs Manual, または emacs の Info を見て下さい.

4.2.5 mh-e の使い方

準備

mh-e を使ってメールを読み書きするためには最小限, 以下の準備が必要です. 即ち, `~/.mh_profile` というファイルを作成し,

```
Path: Mail
```

と書いておきます. これは読んだメールを保存するディレクトリとして, `~/Mail` を指定するということです. 指定したディレクトリが無ければ mh が作成してくれます.

メールを送る

emacs を起動し, `M-x mh-smail` とすると, ミニバッファに `To:`, `Cc:`, `Subject:` がプロンプト (入力要求) されますから入力します. 意味は順に, 宛先, カーボンコピーの送り先, 題目, です. カーボンコピーを送りたくなければ, `Cc:` のところではただリターンを入力すればいいです.
これらを入力し終わると, draft-?? というバッファが開きます. 先に入力した宛先や題目が表示されているので, その下にメールの内容を書きます. 宛先や題目が間違っていたら, ここで書き直しても構いません.
メッセージを書き終わったら, `C-c C-c` としてメールを送ります. メールの最後にいつも自分の名前やアドレスなどを書くのが面倒臭かったら, ホームディレクトリーに `.signature` というファイルを用意し, その中にアドレスなどを書いておきます. そして, メールを送り出す前に `C-c C-s` とすると, `.signature` の中身がメールに挿入されます.

メールを読む

geoph などに login したとき mail が来ているのであれば “You have new mail” と出ます.
emacs を起動し, `M-x mh-rmail` とすると,

4.2　電子メールの使い方

4.2.1　電子メールの仕組み

電子メールの仕組みを簡単に述べます. すなわち, 電子メールはどのようにして相手に届くかを示します.

電子メールを投函すると先ず, 投函された計算機に一時貯められて, 投函された計算機の郵便配達の為のプログラムが処理をします.

このプログラムは電子メールアドレスを見て, それにしたがって投函された手紙を他の計算機に渡します.

渡された計算機の郵便配達の為のプログラムは, まず, 一旦, その手紙をディスクに貯めます.

もし, 渡された電子メールの宛先が自分ではなければ, その宛先にしたがって, また, 別の計算機に渡します.

渡された電子メールの宛先が自分であれば, そこで各個人の郵便受けに入れます. これは/var/spool/mail という場所です. ここに各個人の郵便受けが並んでいます.

これを繰り返して電子メールは届くのです.

電子メールが届いたら, ファイルの先頭の方を見て下さい. 計算機間のやり取りの記録が付いています.

4.2.2　電子メールアドレス

電子メールの宛先はそれぞれの計算機の管理区域によって決められています.

地球惑星物理学科の計算機の電子メールアドレスは

loginname@geoph.s.u-tokyo.ac.jp

です. 例えば s22516 の電子メールアドレスは

s22516@geoph.s.u-tokyo.ac.jp

です.

誰かに手紙を出したり, 送ったりする前に, まず, 自分に手紙を送ってみましょう. 確かに届くか確認して下さい.

4.2.3　電子メールを使う時の注意

電子メールには郵政省の物理メールを使う時と同様にマナーというものがあります. もちろん, 文の内容などは日常使っている手紙と同様のマナーで良いでしょう.

subject

Subject には今の所漢字やカタカナは使えません. また, 全角の英数字も駄目です.

手紙には署名を付けましょう.

手紙の末尾に数行の自分を表す署名をつけます. いろいろと凝った署名をつけている人もいます. 個人のセンスを発揮して下さい. ただ, 世の中には悪趣味という言葉もあるのを忘れずに. 特に相手が, 例えば 80×25 の端末で読んでいるかも知れない時に長い署名 (例えば 5 行以上) をつけていると, 相手に迷惑をかけるかも知れないということをお忘れなく.

```
tansei1.tansei.cc.u-tokyo.ac.jp
space.mit.edu
geophys.washington.edu
```

各名前はなにかしらのアルファベットが . (ドット) によって区切られた形からなっていることが分かるだろう. このドットに区切られた各部分をラベルと呼ぶ.

各名前を後ろから見てみよう. まず, jp とは japan つまり国別コードを表している. 歴史的な理由からアメリカ合州国を除き, 全ての国の計算機の名前は国別コードからはじまっている. ここを top domain と呼ぶこともある. 最後の二つの, edu で終っているのはアメリカ合州国の計算機である. 以下, 上の四つの名前についてみていこう.

次は ac. これは academy の略. つまり大学や (主に国立の) 研究施設を指す.

次の u-tokyo は東京大学を指している.

次の s は理学部, ecc は教育用計算機センター (educational computer center), eri は地震研究所, ccsr は気候システム研究センター, cc は大型計算機センターを指している.

次を見てみよう, eccs01-s, hato, climate1 はもう最後のラベルになっているが, geoph, tansei はまだ下にラベルがある. 我々はこのことから名前は任意の個数のラベルからなることが分かる. geoph はもちろん我が地球惑星物理学科の意味である.

通常, この二番目から最後のラベルまでをサブドメイン名と呼んでいる. また, 最初の名前を (狭義の) マシン名と呼ぶ. つまり, eccs01-s, hato, climate1, tansei1 はマシン名である. 注意しておきたいのはドメイン名の規則のみを使ってマシンの名前とサブドメイン名の名前を区別することはできないことである.

憶えておくことは, 正式な名前というのは, それは国別コードまでを含んだものであるということである. それ以外のあらゆる省略というものは各地域の設定によるものであり, どこまでの省略が可能かということについては一般的な規則はないということである.

また, 電子メールアドレスとマシンの名前とは一般的には関係が無い.

なお, 地球惑星物理学科の計算機はすべて

```
geoph.s.u-tokyo.ac.jp
```

というサブドメイン名を持ち, 教育用計算機センターの計算機は全て

```
ecc.u-tokyo.ac.jp
```

というサブドメイン名をもつ.

すなわち,

```
geoph.geoph.s.u-tokyo.ac.jp
geopmars.geoph.s.u-tokyo.ac.jp
gpsun01.geoph.s.u-tokyo.ac.jp
```

や

```
eccs0101.ecc.u-tokyo.ac.jp
eccs0102.ecc.u-tokyo.ac.jp
ecc-s4.ecc.u-tokyo.ac.jp
tecc.ecc.u-tokyo.ac.jp
```

などとなるわけである.

第 4 章

計算機ネットワーク

いくつかの標準化され, 交換可能なプログラム単位, あるいはモデュールが, 組み合わされてさまざまな仕事に対応する. 人間を支配する中央機関としてのコンピュータではなく, 記憶を操作する補助装置としてのコンピュータ, 個人と対話するだけでなく, 標準化されているからいたるところに分散しつつ, 個人の間に交信可能な関係の網を織りなしていくコンピュータがここにある.

高橋悠治 *"Asia through Computers"*

この章では計算機間の通信について述べる. 現在の計算機は計算機ネットワークなしには存在できない.

UNIX ワークステーションの大部分の利用形態はワークステーションを単体で使うのではなく, 複数台つなげてファイル, プリンターなどを共有して使う形式を採る. このネットワークの利用方法は, 利用者をしてネットワークを意識させしむことなしにネットワークを使うことを可能たらしめている. 例えば教育用計算機センターの Sun はどのマシンを使っても自分の home directory は同じものを使うことができる. これを可能にしているのが計算機ネットワークである.

さらに計算機ネットワークは陽に通信媒体として使用することもできる. すなわち, Network News, 電子メール, talk などである.

ネットワークを介して目の前にある計算機以外の計算機を利用することもまた, 容易である. rsh の所を見て欲しい.

この章では計算機ネットワークに於けるマシンの名前をまず示す. 次に電子メールと Network News について述べ, 最後に rsh 等のアプリケーションについて述べる.

4.1 Domain Name System

計算機ネットワークに継っている計算機の名前の付け方 (個々の計算機の識別法) については, ネットワークの物理的なレベルからアプリケーションプログラムのレベルまでその抽象度によって方法がちがうが, ここでは通常我々が使っているドメイン名システム (domain name system) について簡単に説明する.

ネットワークに継っている個々の計算機は一つ一つに名前を持っている. ネットワークが無矛盾であるためには, 1 つの名前に対してたかだか 1 台の計算機が対応していなければならない. もし, 1 つの名前に対して 2 台以上の計算機が対応していれば, どの計算機が対象になるかが非決定的になるわけである.

ドメイン名システムに於ける計算機の名前は, 国際郵便の宛先に似ている. 例を見て見よう.

```
geoph.geoph.s.u-tokyo.ac.jp
eccs01-s.ecc.u-tokyo.ac.jp
hato.eri.u-tokyo.ac.jp
climate1.ccsr.u-tokyo.ac.jp
```

せん. しかし, そこに書いてあるのをまねしてフォントをちょっと変えてみたり, スクロールバーをつけないようにしたりするのは簡単ですから興味のある人はトライしてみてください.

.xinitrc ファイル (.xsession ファイル)

このファイルは xinit などとして X-Window を起動した時に実行されます.

機能は大きく分けて 2 つで, 1 つは X-Window を立ち上げた時に必要な設定. もう 1 つは初期画面にどんなプログラムを配置するかなどの決定です. このファイルを書き換えれば自分の好きな画面にすることができます.

.xsession ファイルは X 端末からログインした時に実行されます.

.twmrc ファイル

このファイルは前述した twm というウィンドウ・マネージャの初期設定ファイルです. X-Window でのカスタマイズは実際にはほとんどこのファイルで行ないます. 例えば, マウスのボタンを押したときにどのメニューが表示されるかとか, メニューの中身を変更することもできます. また, ウィンドウのタイトルに使うフォントや, タイトル領域にあるボタンの形を変えたり, 自分の環境の見栄えにうるさい人は必ずいじる必要のあるファイルだと言えるでしょう.

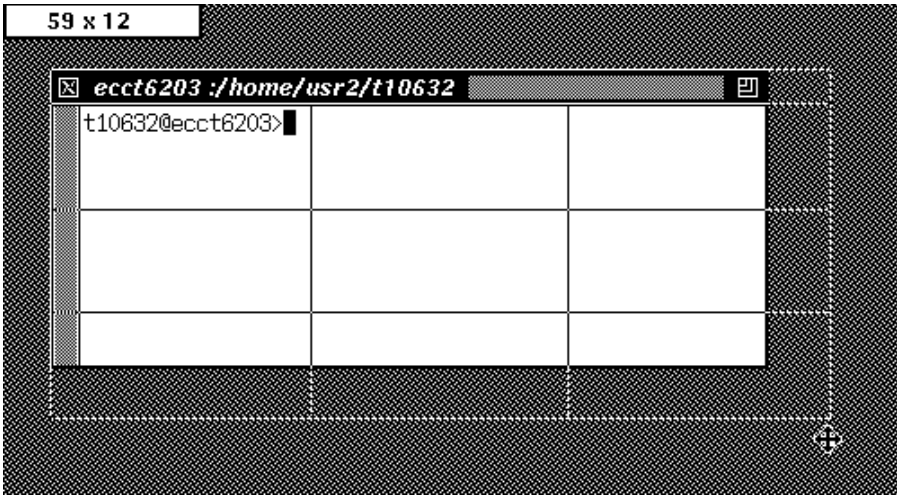


図 3.5: ウインドウの resize

では, タイトルの右ボタンにマウスのカーソルを合わせて, ドラッグして下さい. ウインドウの枠が変形するはずです. 画面の左上にウインドウのサイズが表示されます. この機能を resize と呼びます.

まとめ

以下にマウスのボタンに対する機能を示します. 今まで説明しなかったものも入っています. いろいろお試し下さい.

left title button　: iconify
right title button　: resize

Where	Key	left	middle	right
root		*Twm*	*Clients*	
title bar		move	raise-lower	
Window		select	paste	select
Window	Control	*Main Options*	*VT Options*	*VT Fonts*
Icon		(De)iconify	(De)iconify	

3.1.7　カスタマイズの仕方

X-Window 関係の設定は, ほとんど, `.Xresources(.Xdefaults)`, `.xinitrc(.xsession)`, `.twmrc` という 3 つのファイルで行なっています. ここではそれらがどのようなことを設定しているのかを説明し, 自分の好きなように変更するためのヒントを教えます.

`.Xresources` ファイル (`.Xdefaults` ファイル)

ここでは, 色々なプログラムを実行したときのデフォルトのオプションみたいなものを設定しています. 例えば, 単に `kterm` を起動しただけで必ずスクロールバーをつけるようにするとか, デフォルトでのフォントを指定したりしています. このファイルを書き換えるには多少 X-Window について知らなければなりま

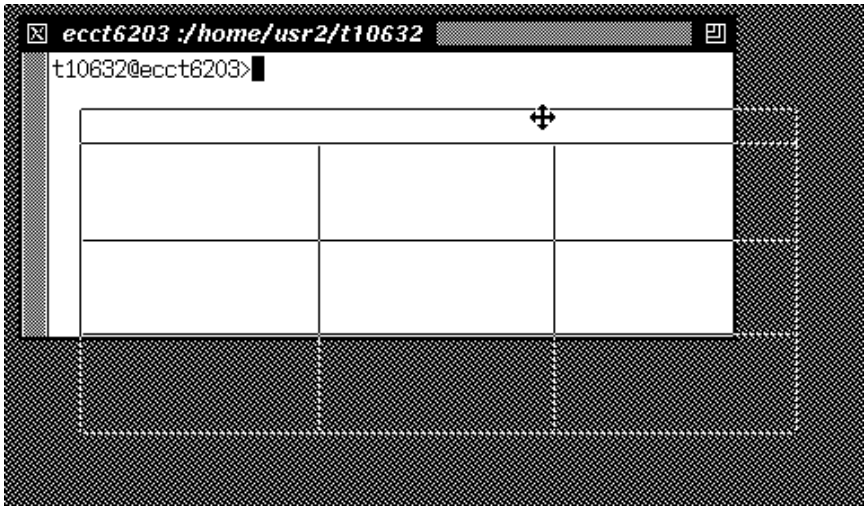


図 3.4: ウインドウの move

3.1.6 ウィンドウの操作

X-Window は言うまでもなくウィンドウ環境ですから, 画面にたくさんのウィンドウを表示しているのが普通です. ここではそれらのウィンドウを動かしたり, 大きさを変えたりといった操作の仕方を説明します.

タイトルバー

初期画面で左下に表示されるウィンドウを見てください. 使っているマシンの名前と現在のディレクトリ名を表示している部分があります. ここの部分をタイトル領域と呼びます (図 3.3). ここにはウィンドウのタイトル以外にもいくつかの物があります. 一番左の X マーク (左タイトルボタン) にマウスカーソルを合わせてマウスのボタン (どのボタンでも構いません) を押してみましょう. ウインドウが消えます. 代わりにの Icon Manager の該当する欄に X のマークが入るか, もしくは小さい箱になります. この操作を iconify と呼びます. また, Icon Manager の各欄, および, 小さい箱のことを icon と呼びます. 逆に, icon にマウスカーソルを持っていって, マウスの左ボタンを押すと, さっきのウインドウが元通り表示されたことと思います. これが De-iconify です.

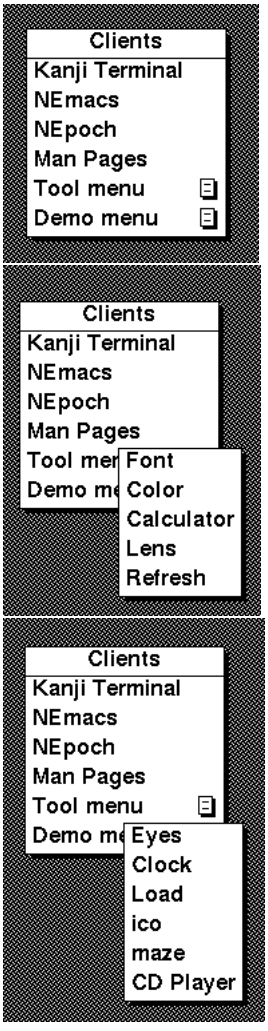
また, ウインドウにマウスカーソルを持っていった時のことを観察して下さい. タイトルバーは灰色になります. また, そのウインドウに文字のカーソルがあればそれが黒くなることが分かるでしょう. この状態でそのウインドウが選択されたことになります. また,Icon Manager の各欄にマウスカーソルを持っていった時もそうなります.

次に, このタイトルの文字の上, または, 灰色になる部分にマウスのカーソルを持っていって下さい.

ここでマウスの真中ボタンを押してみて下さい. もしそのウインドウの一部が別のウインドウに隠れている場合は, 一番手前に, もし, 一番手前にあれば一番後ろにウインドウが行くでしょう. この機能を raise-lower と呼びます.

次に, マウスの左ボタンを押して, 離さずにマウスを移動させて下さい. この, マウスに加える動作をドラッグといいます. ウインドウが動くのが分かるでしょう. ウインドウを動かす機能は move です.

Clients メニュー



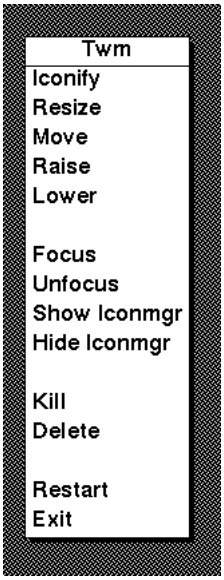
root Window でマウスの真中のボタンを押すと, Clients と書かれたメニューが現われます.

- Kanji Terminal とは kterm (2.2.9 節) を今使っているマシンに開かせるものです. これは一番初めに開いている大きなウインドウと同等のウインドウを開くことになります.
- NEmacs というのはエディタ emacs (2.9 節) を起動します. このエディタは何もプログラムを組むときにしか使わない, といったものではありません. 他人にメールを出すことも, 届いたメールを読むことも, ニュースを読み書きすることもできます.
- NEpoch は emacs の改良版エディタ epoch を起動します. 詳しいことは emacs 上の info (2.7.2 節) を見て下さい.
- Man Pages は X-window 上でのマニュアル xman を起動します. manual については 2.7.1 節 を参考にして下さい.
- Tool menu にはサブメニューがあって, いろいろ便利なツールが入っています.
  - Font では font 見本を見ることができます. xfontsel が起動されています
  - Color では色見本を見ることができます. xcolors が起動されています.
  - Calculator は画面上の電卓です. 関数電卓になっています. 終了は AC をマウスの右ボタンでクリックして下さい. xcalc が起動されています.
  - Lens は画面の虫メガネです. マウスの左ボタンをクリックしたところが大きくなります. xmag が起動されています.
  - Refresh は画面全体の描き直しです. 何らかの理由で画面がおかしくなったときにこれを使います. xrefresh が起動されています.
- Demo menu にはサブメニューがあって, 「おまけ」的なツールが入っています.
  - Eyes, Clock, ico, maze については特に説明することはありません. まあやってみて下さい.
  - Load はシステムの平均的な負荷状態をグラフにするものです.
  - CD Player は CD プレイヤーがつながっている場合にそれを動かすためのものです.

- Icon Manager を表示するかどうかの切替が, `Show Iconmgr` および `Hide Iconmgr` です.
- `Kill` および `Delete` はウインドウを消すためのものです. 選択すると, カーソルが髑髏マークになります. それを消したいウインドウの上に持っていってクリックすると (どのボタンでも構いません), そのウインドウが消えます. kill と delete の違いは, kill の方が強制的な消し方です. 従って, 何か消したい時は, delete でためしてみて, それで駄目だったら, kill をやるというふうにするのが安全です.
- `Restart` は twm の再起動をします.
- `Exit` は twm の終了です. 地物標準環境では, X system 全体も同時に終了します.

Twm メニュー

root Window でマウスの左のボタンを押すと, Twm と書かれたメニューが現われます.



- Iconify を選択すると, カーソルの形が変わります. そのカーソルを, どこかのウインドウに持っていって, そこでマウスをクリックすると (どのボタンでも構いません), そのウインドウが消えて, Icon Manager の該当する欄に X のマークが入るか, もしくは小さい箱になります. この操作を iconify といいます. また, 小さい箱のことを icon といいます. Icon Manager の各欄も icon といいます. これと同じことは, タイトルバーの左にある黒丸マーク (左タイトルボタン) をクリックすることによってもできます (3.1.6 節). 元へ戻すには, icon のところでマウスの左ボタンを押します. この操作を De-iconify といいます.

- Resize を選択すると, カーソルの形が変わります. そのカーソルを, どこかのウインドウに持っていって, そこでマウスのボタン (どのボタンでも構いません) を押し下げたまま枠のところまで動かしていくと (ドラッグしていくと), 枠の大きさがマウスの動きにしたがって変わります. この機能が resize です. 同じことは, 右タイトルボタンを使ってもできます (3.1.6 節).

- Move を選択すると, カーソルの形が変わります. そのカーソルを, どこかのウインドウに持っていって, そこでマウスのボタン (どのボタンでも構いません) を押し下げたまま動かすと (ドラッグ), マウスの動きにしたがって, ウインドウが動きます. この機能を move といいます. 同じことは, タイトルバーをマウスの左ボタンでドラッグしてもできます (3.1.6 節).

- Raise を選択すると, カーソルの形が変わります. そのカーソルを, どこかのウインドウに持っていって, そこでマウスをクリックすると (どのボタンでも構いません), そのウインドウが他のウインドウよりも前に来ます. この機能を raise といいます.

- Lower を選択すると, カーソルの形が変わります. そのカーソルを, どこかのウインドウに持っていって, そこでマウスをクリックすると (どのボタンでも構いません), そのウインドウが他のウインドウよりも前に来ます. この機能を lower といいます. raise と lower を組み合わせた raise-lower という機能はタイトルバーをマウスの真中ボタンをクリックすることによってできます (3.1.6 節).

- Focus を選択すると, マウスカーソルが黒丸に変わります. そのカーソルを, どこかのウインドウに持っていって, そこでマウスをクリックします (どのボタンでも構いません). 以後, マウスカーソルがどこにあっても, キーボードからの入力はすべてそのウィンドウに入力されます. この機能が focus です.

- focus を解除するには Unfocus を選択します.

3.1.5 root Window でマウスのボタンを押してみると...

ここでは twm というウインドウマネージャーの説明をします. ウインドウマネージャーというのは個々のウインドウの位置決めをしたり, どのウインドウにキー入力を割り振ったりするかということを世話してくれるプログラムです. X-Window での操作は基本的にマウスを使いますが, どのボタンをどこで押すかによって動作が異なります. ここではどこでどのボタンを押すとどうなるのかを説明します. これもユーザーが自由に設定できるので, 他の人の環境では全く違う機能が割つけられているかも知れません. ともあれ, 地物環境の設定を説明します.

図 3.3を見てください. 画面の地の部分を root window と呼びます. この画面の root window でマウスのボタンを押してみます. マウスのボタンを押したままマウスを動かす[4]と各項目が強調表示されますので, 好きな項目を強調表示にします. その状態でマウスのボタンを離せばその項目を選択出来るというわけです.

[4] この操作をドラッグと呼びます.

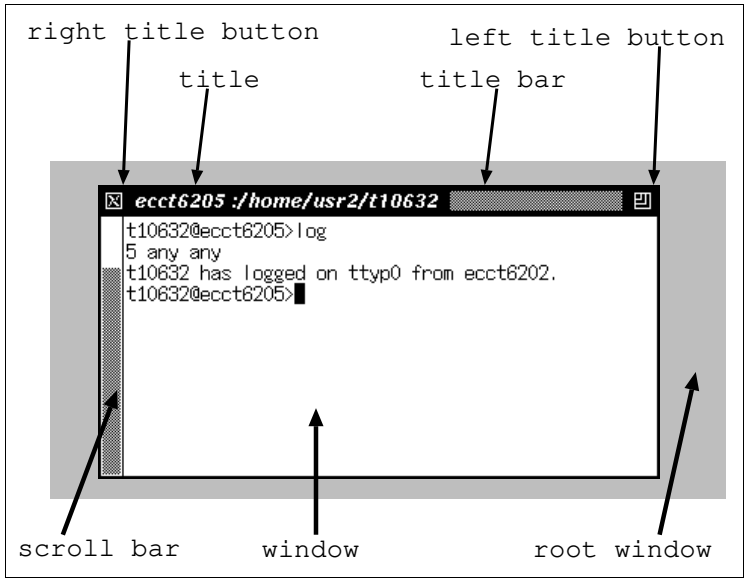


図 3.3: kterm と title bar

右下の郵便受けは, メールが新しく届いたことを知らせるものです. 誰かからメールが来ると色が反転して旗が上がります. プログラム名は xbiff です.

もう一度左上に戻ると, TWM icon manager というタイトルがついたウインドウが表示されています. 下に, ウインドウの名前がいくつか表示されています. これは Icon Manager と言うもので, ここをマウスの左ボタンでクリックする[2] ことで好きなウィンドウを表示させたり, 一時的に退避させたりすることができます. また, マウスの中ボタンをクリックするとそのウインドウを一番手前に持ってくることができます. 加えて, マウスのカーソルがここにある時, その対応するウインドウに対してキー入力を受けつける状態になっています.

右下にある Console という名前がついたウインドウは他のウインドウとはちょっと違う特別な意味を持っています. このウィンドウではコマンドを入力することもできますが, 小さいので結果を見るのには不便でしょう. システムからのメッセージ等はこのウィンドウに表示されますので, そのようなメッセージの表示用と考えてください[3].

右上の emacs エディタについてはすでに 2.2.11, 2.9 節で説明しました.

左下の大きなウィンドウは, 今使用しているマシンの名前が title についているはずです. このウインドウは *kterm* というプログラムのウインドウです. *kterm* は疑似端末で, ここから計算機の命令を入力することができます. *kterm* は kanji terminal emulator の名前で, 上の login ウインドウに使っている *xterm* を漢字が出せるように改造したものです.

ウィンドウには左端に棒のようなものがありますね. これはスクロールバーと呼ばれるものです. ここにマウスカーソルを持っていけば, ウィンドウの中身をスクロールさせて前に実行した結果などをある程度さかのぼって見ることができます. スクロールバーでマウスの右ボタンを押せば下向きに, 左ボタンを押せば上向きにスクロールします. また, 真中のボタンを押しながらマウスカーソルを上下に移動させると, それにつれてウィンドウの中身もスクロールします. やってみればすぐにわかるでしょう.

[2] マウスのボタンを押してすぐ離すこと.
[3] もちろん, これから先に述べるような方法でこのウィンドウを大きくすることもできます.

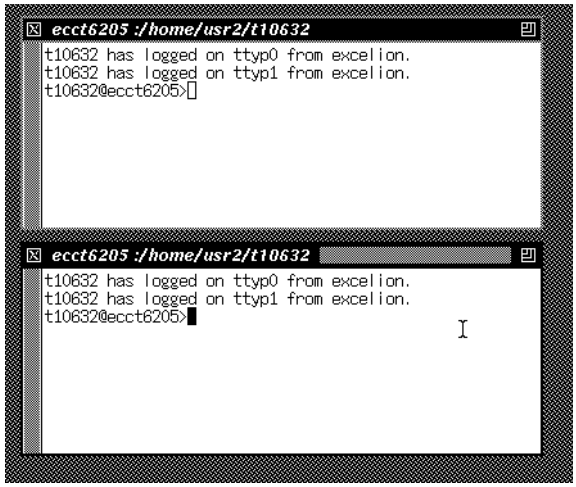


図 3.2: 選択された状態

3.1.2 X-Window の終了

X-window を終る方法は 2.2.8 節で説明しました. ここで, もう一度読み直して下さい. root window でマウスの左ボタンを押し, Exit を選択するのでした.

ここで気をつけるべきことは, まず, 画面に出ている emacs やその他の何かを作っているプログラムを終了させることです. X-window が終了すると強制的に emacs も終了させられるのでせっかく作ったファイルを失ってしまいます.

また, X-window がうまく実行しているプログラムを殺せない時にはそのプログラムのプロセスが残って, 最悪の時には暴走してしまいますのでゲームなどはきちんと終了させた方が良いでしょう.

3.1.3 twm

twm とは Tab Window Manager の頭文字を取ったものと正式にはされていますが, Tom LaStrange という人が作ったので twm といわれていると一般には信じられています. Window Manager とはウインドウの位置やウインドウの状態, キー入力の割り振り等を管理するプログラムです.

ウインドウの位置などの操作は後に譲ります.

まず, キー入力の割り振りについて述べます. キー入力は基本的にマウスのカーソルがあるウインドウに割り振られます. twm ではキー入力を割り振られたときにタイトルを反転します. また, 枠の色も変わります. 図 3.2を参照して下さい. 上が選択されていない状態, 下が "I" 形のマウスカーソルがウインドウの中にあって選択された状態の *kterm* です.

X-window System の各設定はユーザーによってかなり変えられます. 設定法はオンラインマニュアルに譲ります. ここではとりあえず地物標準環境で設定したものの説明をします.

3.1.4 初期画面の説明

`xinit` でX-Window を起動すると, 左上に時計と twm icon manager, 右下に郵便受けと Console window, 左下に 1 つの大きなウィンドウ, 右上に大きく emacs が表示されます.

左上の時計は説明の必要もないでしょう. これは `xclock` というプログラムです.

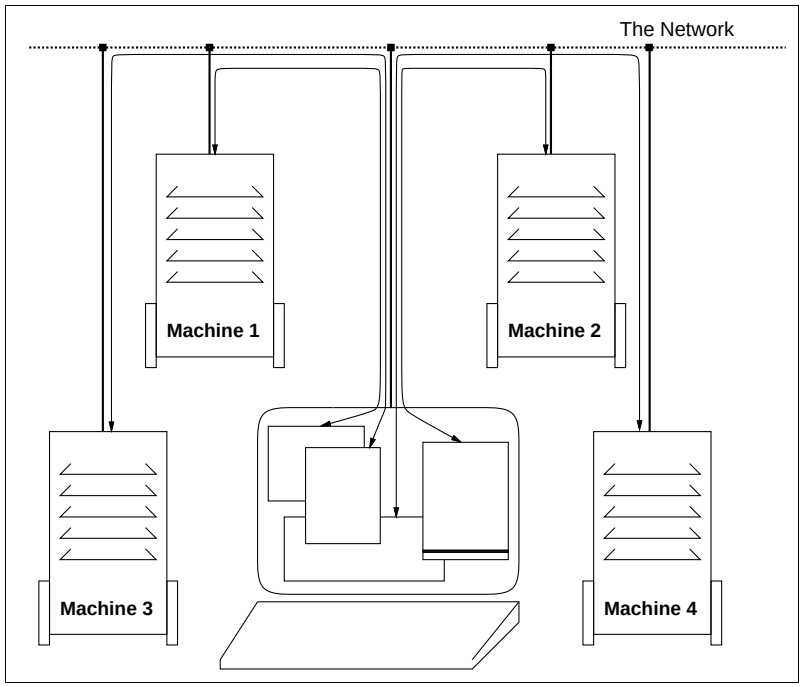


図 3.1: X-window とネットワーク環境

第 3 章

ウィンドウ ・システム

> ファンアイクの名高いデッサン 聖バルバラ」(1438 年) は, 巨大なゴシックの塔を背にして端然とすわっている聖女を描いたものだ. 伝説によると, 彼女はヘリオポリスの富豪の家に生まれたが, 娘の結婚を許さない嫉妬深い父親のため, 高い塔の内部に幽閉された. 孤独のなかで, やがてキリスト教に改宗した彼女は, ある日, 父の不在中, それまで二つしか窓のなかった塔の壁に, 職人に命じて三つ目の窓を穿たせた. 魂は三つの窓, 即ち聖なる三位一体によって光明を受けねばならないからである. これを聞いて激怒した異教徒の父親は, みずから娘の首を刎ねたという.
>
> 澁澤龍彦 「少女コレクション序説」

ウインドウシステムとは, 画面上にいくつもの領域=ウインドウを設定して複数のプログラムの出力をそれぞれに表示させるのです.

プログラム作成一つとっても, エディタでプログラムを編集して, それをコンパイルして, 実行しながらデバッグするといった作業があります. また, 一つのウインドウでドキュメントを表示させながらもう一つでプログラムを書くことができ, これらをウィンドウ環境で並行して進めるのと一つの画面で順番にやるのとでは能率がちがいます.

3.1 X-Window

Unix workstation をここまで普及させた要因の一つに Window 環境があります. 特に X-window というウインドウシステムは様々な機種用に作られ, それらが同一のプロトコル[1] を持っていて, 他のマシンの上で走っているクライアントプログラムが今自分が使っているマシンの画面に自由にアクセスできるので, (図 3.1 参照) Unix ワークステーションの事実上の標準になりました.

3.1.1 X-window の起動

X-window の起動は次のようにします. geoph に login していて, まだ X-window を起動していない状態で

```
xinit
```

という命令を実行します. 既にウインドウシステムを起動している場合には決してさらに X-window を起動してはなりません.

[1] ソフトウエアーどうしの通信規格のことです.

```
boiled-egg habenrida
```

の直後に C-j を打つと

```
boiled-egg | 葉 便利だ |
```

の様になる. 後は, C-p で 「葉」を 「は」に変えてリターンキーを押せば

boiled-egg は便利だ

の様になる. というわけで, アルファベットの直前直後にはいつでも空白を空けておく習慣にしておくと, アルファベットと日本語が混ざった文章を書くのにいちいち C-\ でモードを変えなくて済む.

boiled-egg を使うには, 文章を書いているバッファで, 一度

```
M-x boil
```

とやれば良く, その時から後はそのバッファではずっと boiled-egg が使える. ただし, 標準環境では TEX モードにおいては, デフォルトで boiled-egg が使えるようになっている.

2.9.16　その他

フェンスモードで, M-k とすると, フェンスの中が全てカタカナになります.

日本語のコード体系には, 主に, JIS, Shift-JIS, EUC があります.

emacs の中で, 漢字コードを変えるのには, C-x C-k f を用います. (egg-sun-keymap を load していれば, PF1 で切り換えできます.)

emacs の外で漢字コードを変えるには nkf を用います.

Egg の辞書登録は, リージョンを設定した後, M-x toroku-region とします.

- 文節の長さを 1 文字短くする:C-i
- 全部確定:リターンキー

一つ日本語変換の実例をあげておきましょう.

「私は教育用計算機センターの指導員室にいます.」という文を作りたいとします.

まず, Ctrl-\ と打って日本語モードにしましょう.

次に, ローマ字で文を入力します.

watasihakyouikuyoukeisankisenta-no このへんで切りましょうか.

長音はハイフンを用います. ローマ字は自動的に仮名に変換されていきます.

| わたしはきょういくようけいさんきせんたーの |

のように, 両端に | | がついています. この | をフェンスと呼び, フェンスではさまれた部分が変換の対象になっていることを示します.

ここで, スペースバーを押すと, 漢字に変換されます. ところが, この日本語変換システム (egg という) は, たまに (or よく) 妙な変換をします. 例えば, 次のようになったとしましょう. [18]

| 私は 驚異 供養 計算機 センターの |

文節の区切りのところには小さなスペースが入っています. カーソルは, 初めの 「私」のところにあります.

「きょういくよう」のところが誤っていますので, C-f でカーソルを右に 1 文節移動します (「驚」のところに来る). そして, 「きょうい」を 「きょういく」にするには, 文節の区切りを右に 1 文字分伸ばすことになりますので, C-o と打ちます. すると,

| 私は 教育 洋 計算機 センターの |

のようになるでしょう.

あとは, またカーソルを右に移動し, スペースバーを何回か押して 「洋」を正しく 「用」に直します. そしてリターンキーを押せば全文が確定します.

続けて後半も入力します. 句点 (.) はピリオド, 読点 (,) はカンマを入力すれば, それぞれ変換されます.

| しどういんしつにいます. |

ここでスペースを押すと,

| 指導 陰湿に います. |

なんてなるかもしれません. 試しに, C-o を 2 回打ってみましょう. 「指導員」となればめでたしですが, もしだめなら, もう 2 回 C-i を打って元に戻し, 「指導」のままにします. そして, C-f で次の文節に移ります. まさか 「員室」などという語はないでしょうから, 「員」「室」の 2 語に分けます. つまり, C-i を 3 回ということになります.

また, ある文節をカタカナに変換したいときは, とりあえずスペースを打って漢字に直し, それから C-p を 2 回打ちます.

boiled-egg による日本語変換

geoph においては boiled-egg というソフトウエアが用意されていて, 特にアルファベットと日本語が混ざったような文を書く場合には便利に使える.

標準の egg においては日本語モードとアルファベットモードとを変える時にその都度 C-\ を入力しないといけないのが煩わしい. boiled-egg では, C-j によって前のスペースからその直前の場所までが漢字に変換される. 例えば,

[18] 辞書は学習しますから, 常に同じになるとは限りません.

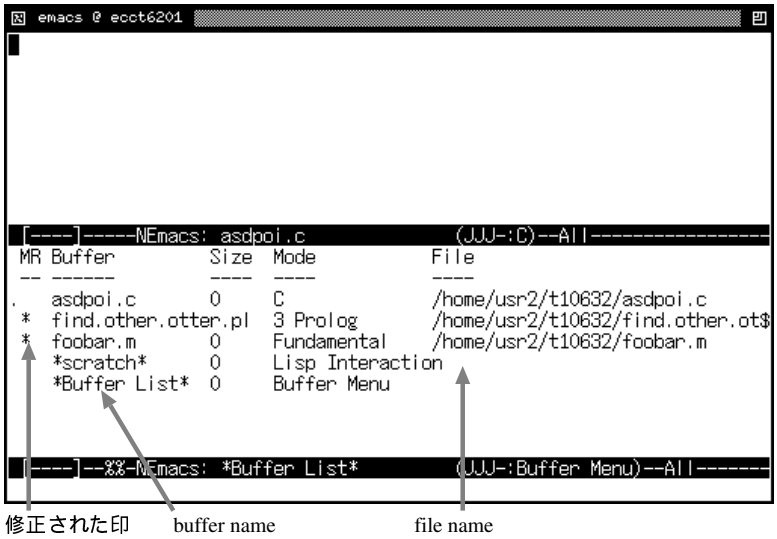


図 2.2: emacs のバッファリスト

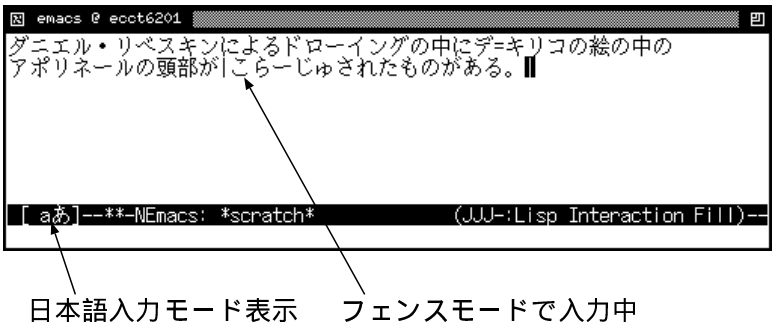


図 2.3: emacs で漢字を入力

2.9.15　日本語変換の方法

まず, C-\ と打つと日本語ローマ字入力モードになります. (画面の左下に [a あ] と表示が出ます). もう 1 回打つと英数字モードに戻ります.

複文節変換ができるので, 適当なところで区切って変換します. 変換するにはスペースバーを押してください.

変換するモードにはいると画面の左下の [a あ] が [漢字] と変わります. ここで以下のキーが使えます.

- 次の変換候補:C-n またはスペースバー
- 前の変換候補:C-p
- 右の文節に移る:C-f
- 左の文節に移る:C-b
- 文節の長さを 1 文字長くする:C-o

文字の削除

文字の削除はいろいろな命令が揃っています.

- カーソルの位置にある 1 文字の削除: C-d (delete); カーソル位置はそのまま.
- カーソルの左にある 1 文字の削除:バックスペースキー (⇐) ; カーソルは 1 つ左に進みます.
- カーソルより右側を 1 行分削除: C-k
- 範囲を指定して削除, 移動, または複写:

  1. 指定する範囲の最初の文字にカーソルを移動し, C-@ またはC-(SPC)(コントロール空白)
  2. 指定する範囲の最後の文字の次にカーソルを移動し, C-w
  3. その範囲を削除するだけならこれで終わりです.
  4. 指定した範囲をよそに移動する場合は, 持っていきたい箇所にカーソルを移動し, C-y とすると, 今削除した範囲がそこに挿入されます. C-y は, 最も最近に削除した範囲をテキストに挿入するコマンドです. これをヤンクと呼びます. .
  5. 指定した範囲を削除せずに, よそにも複写する場合は, (2) が終わったところで一旦 C-y とすると, もとのところに文章が復活するので, それから複写先にカーソルを移動して, 再び C-y

C-(SPC) を打った時にエコー領域に Mark set と表示されます. そこに現在のマークがつくのです. このマークが付いたところからポイント迄をリージョン (region) と呼びます.

2.9.12 再表示

画面が乱れたとき: C-l(エル) で再表示されます.

2.9.13 バッファの切り替え

emacs はたくさんのファイルを扱うことができます. C-x C-b で図 2.2のような一覧表が出てきます. また, 操作しているバッファを換えるには C-x b を使います.

2.9.14 複数のウインドウ

複数のウインドウを開けて操作をすることは, 編集作業を格段に楽にします. 例えば同じプログラムの上の方に書いた定義を読みながら下の方でそれを使うとか, 違うプログラムを参考にしながらプログラムを書き, 必要な所は持ってきたりするなどといった作業が可能になります.

- ウインドウを上下 2 つに分割:C-x 2
- 他のウインドウに移る:C-x o
- 現在ポイントがあるウインドウのみを表示:C-x 1(“1” とは一です.)
- 現在ポイントがあるウインドウを非表示にする:C-x 0 (“0” とはゼロです.)

2.9.8　命令の中止

命令を違えたとき

```
C-g
```

で中止できます. ただし既にその命令が実行された後ではこれでは元には戻れません. 上の, ファイルの読み込みでファイル名を聞いてきた時に C-g を実行してみましょう. Quit とミニバッファに表示され上の命令が中止されるでしょう.

2.9.9　カーソルの移動方法

まずは基本的な 1 文字の移動を示します.

- 右に移動:C-f(CTRL キーを押しながら f と打つこと. 以下同様)f は forward の略.
- 左に移動:C-b(backward)
- 上に移動:C-p(previous line)
- 下に移動:C-n(next line)

このキー配列はこの後いろいろなプログラムで使用されますから「体で覚える」くらいに慣れることになるでしょう. ただし, テンキーも使用可能です.

さすがにこれだけでは大きなファイルを処理するのは大変なので色々な, 移動するための命令があります.

- 行の左端に移動:C-a
- 行の右端に移動:C-e
- 文書の頭に移動:M-<(ESC キーを打ってから<と打つこと)
- 文書の末尾に移動:M->
- 1 画面分下へ移動:C-v
- 1 画面分上へ移動:M-v

さらに, 後に説明する検索機能を使って効率良く移動ができるようになっています.

2.9.10　文字列の検索

文字列の検索には何種類かの方法がありますが, 一般に使われる方法はインクリメンタルサーチです.

カーソルがある場所から, ファイルの先頭に向かって検索するのが isearch-backword C-r, その逆が isearch-forward C-s です. どちらも検索する文字を入れる毎にその文字列にマッチするもののところにポイントを移動します. 連続して次のマッチする場所に行くのは続けて C-r なり C-s を押すことによって実現できます.

2.9.11　編集

文字を入れる

文字の挿入は, 該当箇所にカーソルを移動し, 挿入したい文字をタイプするだけでよいです. これは改めて書くまでもないと思う人もいるかも知れないですが, 例えば vi (別のエディタです.) はそうではありません.

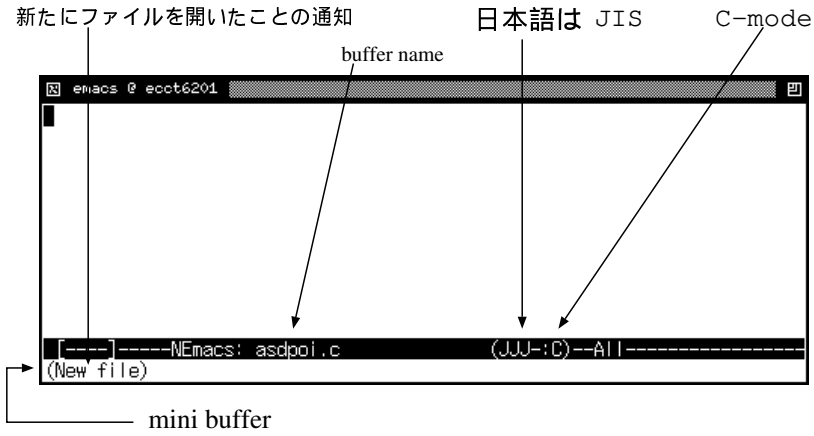


図 2.1: emacs で新しいファイルを読み込む. 及び各部の説明

2.9.5 ファイルのセーヴ

ファイルのセーヴ[17]は

```
C-x C-s
```

です. s は save の s です.

ファイルのセーヴは emacs を終了する時のみにするものではありません. いつ何時 emacs や使っているマシンが落ちても良いように, 編集の途中でもこまめにセーヴしておきましょう. こうすることによって助かることが良くあります.

2.9.6 ファイルの読み込み

ファイルを読み込むには

```
C-x C-f
```

です. f は find-file の f です.

このコマンドを実行したら emacs はミニバッファでファイル名を聞いてきます. それに答えたら今まで編集していたファイルとは違うバッファを新たに開き, 存在するファイル名を答えたのであれば, そのファイルを読み込み, 存在しないファイル名を答えたのであればそのファイル名のファイル用のバッファとなります.

通常新たにファイルを作る時はこうして作ります.

2.9.7 今いるバッファにファイルを挿入する

今いるバッファのカーソルにファイルを挿入するのは

```
C-x i
```

です. ミニバッファでファイル名を聞いて来るので, 入力します.

[17] ファイルの内容を保存すること.

2.9.2　概念

emacs で使われている概念を以下に示します.

バッファ ファイルを読み込んだ時に, そのファイルを計算機の記憶に保存しておくための領域のこと. emacs を終了するとその内容は消えてなくなる. バッファには読み込んだファイルの名前が付けられる.

ウィンドウ emacs にファイルを読み込んだ時にファイルの中身が表示される部分のこと. 実際に emacs を立ち上げている状態では, 下の方に黒いバー (次に示すモードラインのこと) があり, その上がウィンドウになっている. また, emacs ではウィンドウをいくつにも分けて, 異なるバッファの内容や同じバッファの異なる部分を表示することができる. (X-window のウインドウとは違います.)

モードライン バッファの状態を示している行のこと. 実際には emacs の画面で下の方に引かれた (ウィンドウをいくつかに分割した場合には, 各ウィンドウの下に引かれる) 黒いバーの部分を指す.

エコー領域 emacs からのメッセージなどを表示する行のこと. モードラインの下の部分を指す.

ミニバッファ M-x 命令名のように, emacs に利用者が指示をする時に用いられるバッファ(実際には, このミニバッファはエコー領域でもある).

モード 現在編集中のファイルの状態を示すもの.

カーソル 編集コマンドが効果を及ぼす位置を示すもの (画面にただ一つ存在する).

ポイント 編集コマンドが効果を及ぼす位置 (カーソルが文字の上にある時には, ポイントはその文字とその前の文字との間にあると考える).

2.9.3　emacs の起動

emacs は複数のファイルを同時に編集できます. emacs はとても大きなプログラムで, ワークステーションでもあまり多くの emacs を同時には走らせられません. 特に起動に多くの計算機のパワーを使います. emacs の標準的な使い方は, 計算機への login したら 1 回だけ立ち上げ, logout の時まで終了しない使い方です. そうすれば emacs を何回も立ち上げ直さずに済みます. また, 複数の人間で使っている計算機で emacs を一人の人間が幾つも立ち上げていることは非難されるに値します.

emacs を起動する方法を示します. 他のプログラムと同じく, shell の入力待ち状態で

```
emacs
```

とタイプして下さい. X-window を使っている時には, emacs は専用のウインドウを空けますから,

```
emacs &
```

と back ground process として走らせると, 先ほど言った使い方が簡単にできます.

2.9.4　emacs の終了

終了は,

```
C-x C-c
```

と打ちます. emacs が終了すると今まで emacs が編集していたものは計算機のメモリからは消えてなくなりますので, 終了する前にファイルはきちんとセーヴしなければなりません. 編集した内容を保存していないと,　「本当に終了してよいか」と聞いてくるので, よければ yes と答えて下さい. 保存してある場合や, ファイルの内容に手を加えなかった場合は, 聞いてこずにそのまま終了します.

lesskey

less のキー割当を変更するためのプログラムです.
マニュアルは sun の /usr/local/pub/man にありますので, man コマンドで見ることができます.
lesskey を実行すると, default では~/.less にその設定が保存されます.

環境変数

less は以下の環境変数を参照します.

LESS	オプションの設定
EDITOR	v コマンドで起動するエディタの設定
SHELL	! コマンドで起動するシェルの設定
LESSCHARSET	漢字コードの指定

またアプリケーションによっては, 環境変数 PAGER を参照するものがありますので (例えば man), less を使用するならば,

```
setenv PAGER less
```

としておくと良いでしょう.

2.8.2 more

more は less よりもやや機能が少ないページャーです. 正確には, less は more の改良版です. more は UNIX には必ず付いていますが, less は管理者がインストールしていない場合もあります. 使い方は, less とほぼ同じです.
more は less 同様に環境変数 MORE と EDITOR を参照します.

2.9 emacs

emacs は Unix についてくるエディタではありません. しかし, 現在事実上の Unix の標準のエディタとなっています. それは, emacs と他のプログラムとの連携がとてもうまく実現されているからです. この章ではこの emacs の使い方の初歩を解説します. また, この本の別の章でも emacs はいろいろ出てきます.
emacs は非常に多機能なエディタであり, 詳しく述べることはできません. 詳しくは, 生協で売っている[16]GNU Emacs マニュアルを見て下さい. また, 日本語入力については Info の EGG マニュアルを見て下さい.

2.9.1 キー操作表記の約束

以下の説明では, 次のような略記を使います.

C-x	コントロールキーを押しながら x を打つこと.
ESC	Esc キーを押すこと.
SPC	SPC キーを打つこと.
M-x	メタキーを押しながら x キーを打つこと. あるいは, Esc キーを押してから, x を打つこと.

[16] 学部学生用には 016 号室に置いてあります.

漢字コード

1995 年 3 月現在の less の version は ver.177+kanji+uncompress,gunzip です. 入出力漢字コードの設定は環境変数 LESSCHARSET で以下のものから 1 つ選べます.

```
ascii, latan1, jis, ujis-jis, sjis-jis, ujis, jis-ujis, sjis-ujis,
sjis, jis-sjis, ujis-sjis, japanese-jis, japanese-ujis, japanese-sjis
```

この "ujis-jis" というキャラクタセットは,ujis と jis という漢字コードを解釈し,jis コードで出力するといった意味です. また漢字コードとして japanese と指定すれば,jis.ujis.sjis という 3 つの漢字コードを全て解釈するようになります (その代り半角片仮名を認識できなくなります).

geoph や教育用計算機センターの less は default が japanese-ujis になっています.

命令

最初に覚えるべきコマンドは "H(help)" です. H と入力すれば簡単な使用法が表示されます.

コマンドは vi に似ています. また Control 文字に対しては emacs と似た体系にもなっています. 以下に基本的なコマンドについて説明します.

移動コマンド

j	一行進む
k	一行戻る
d	半ページ進む
u	半ページ戻る
SPACE,f	一ページ進む
b	一ページ戻る
g	ファイルの先頭に移動する
G	ファイルの末尾に移動する

上のコマンドでほとんどの用は済んでしまいます. それぞれのコマンドは vi のように引数をとります. たとえば 10j とすると 10 行スクロールアップし, 100g とすると 100 行目に移動します.

検索コマンド

/	文字列をファイルの末尾に向かって検索する
?	文字列をファイルの先頭に向かって検索する
n	直前の検索を繰り返す

検索には正規表現を使用することができます. また引数をとることができます.

その他

!	コマンドを実行する
v	エディタを起動する

```
ld, ld.so (1)           - link editor, dynamic link editor
ldconfig (8)            - link-editor configuration
link (2V)               - make a hard link to a file
link (5)                - link editor interfaces
.......
.......
.......
```

の様にキーワードをもつマニュアルエントリを表示してくれます.

環境変数 MANPATH

どこにマニュアルがあるかを示す環境変数です. この環境変数に書かれた path の下の man?にあるマニュアルが検索されます. 標準環境では日本語で書かれたマニュアルページが先に探され, その後に英語で書かれたマニュアルが検索されます.

2.7.2 Info

Info とは Emacs のオンラインドキュメントリーダです.

```
M-x info
```

で起動できます.

項目を見るには, 見たい項目にカーソルを合わせ, m をタイプすると見たい項目が現れます.

u で一つ上の層に, q で元のバッファに戻れます. 原語の Gnu Emacs Manual はこの Info ファイルを出力したものですから, こちらも正確な解説が載っています.

紙に打ち出す時には TEX を使います. texinfo.tex というスタイルファイルを使って TEX にかけます. texindex で索引を作ることを忘れないように. 紙への打ち出し方も info を見れば載っています.

2.8 pager

通常の文書ファイルを見る時は cat *filename* コマンドを使います. ですが, ファイルの中身が長い時は, スクロールして最初の方が画面の外へ消えてしまいます. そこで,1 画面分ずつファイルの表示を見るためのツールがページャーです. geoph や教育用計算機センターの計算機にはいくつかのページャーがインストールされています. そのうちよく使われるのは less と more です.

2.8.1 less

起動

less を起動するには

```
less [オプション] [ファイル名]
```

とします. ファイル名を省略した場合は標準入力から読み込まれます.

終了

less を終了する時には “q(quit)” コマンドを使います.

2. system call　open, read, write などのシステムコール[15]の説明.

3. library functions　言語 C の標準ライブラリや数学のライブラリなどの説明.

4. device interface　ディスクやテープなどディバイスドライバの説明.

5. file format　システムやプログラムで使われる様々なファイルについての説明.

6. games, demo　game や demo の説明.

7. その他　主に troff のマクロの説明と, その他の情報.

8. system　システム管理用の命令についての説明. これらの節番号は例えば ls なら 1 に属していますから, ls(1) というように調べる内容の後に括弧つきの数字として現れることがあります.

節の指定

命令の write とシステムコールの write のように複数の節に同じ名前のものがある時には, 単に

```
% man write
```

では, 例えば, write(1) の説明しか表示されません.

```
geoph:/home2/s22516> man -f write
```

のように -f オプションを使うと write に関するエントリを表示してくれます.

```
write (1)               - write a message to another user
write, writev (2V)      - write output
```

こうして節の番号が分かったら, その節の番号をオプションとして man につけてやります. すなはち,

```
geoph:/home2/s22516> man 1 write
```

のようにします.

keyword

キーワードで検索するときには

```
man -k foobar
```

です. 例えば

```
geoph:/home2/s22516>: man -k link
a.out (5)               - assembler and link editor output format
coff (5)                - common assembler and link editor output
dlopen, dlsym, dlerror, dlclose (3X)    - simple programmatic interface to the d
ynamic linker
```

[15] OS が提供する機能

コマンドライン編集

非常に長い命令を入力していて最初の方に誤りを発見した時に, csh では一旦全てを Back Space で消してから打ち直さねばならない. 勿論 kterm を使っている時にはマウスで切り貼りをすることもできるが, あまり賢明な方法とは言えない. tcsh では現在行の編集とヒストリーの編集が emacs や vi のキー配列で実現されている. 何も指定しなければ emacs と同じキー配列をしている.

ファイル名補完

ファイル名補完と言うのはファイル名を途中まで入力した後に, tab キーを押すと完全なファイル名を自動的に表示してくれる機能である. もし, 候補が 1 つに絞れない時には, C-d を押すことによって候補が表示される.

2.7 man と Info

ここではオンラインドキュメントの簡単な読み方を示します. man, Info は共にプログラムやファイルにとっての一番詳細なドキュメントとなっています. これらを打ち出してまとめて売っているも本も多いです.

また, 紙に打ち出していないので, 計算機を使いながらすぐに読め, かつ, 検索が計算機でできるので非常に便利です.

2.7.1 man

man とは manual のことです. もっと言えば, Unix System のオンラインマニュアルを読むプログラムです.

このマニュアルはプログラム, ファイル形式, そして C のライブラリなどに用意されています. 大部分のプログラムにとってはこのマニュアルは正式なマニュアルで, プログラムのソースコードの次にプログラムの正確な仕様を記述していることが多いです.

マニュアルをいくつか読むとマニュアルの読み方も大体分かってきて斜め読みができるようになるでしょう. つまり, 設定すべき環境変数やファイルについては後ろの方にあるのが普通ですし, 命令行のオプションはアルファベット順に箇条書になっているといった具合です. これらを *less* のサーチなどを使って効率的に探す技法を身につけるとよいでしょう.

man の起動

```
man man
```

を実行してください. man の使い方が表示されるでしょう. また, 紙に出したいときには

```
man -t foobar
```

とすると紙に出ます.

section

Unix Manual Pages には内容に応じて 8 つの節があります.

1. command `/bin/ls`, `/bin/ucb/man` などといった命令の説明.

2.6.5　background job

Unix は, 一人で複数のプロセスを同時に実行できる. ウィンドウを使えばこの機能を非常に安易に使用することができるが, これは世にウィンドウシステムが存在する前から存在していた.

プロセスを background[14] で動かすことによって, shell はそのプロセスの終了を待たずに次の命令待ちとなる. これによって, 例えば, emacs を立ち上げた shell も他の仕事ができるようになる.

また, Sun の Unix では background で動いている job は logout してもプログラムが終了するか, 外部から終了の要求が来るまで, 動き続ける. これは何か時間のかかるプロセスを走らせたい時に, 長時間端末を占有しなくとも良いことを意味する.

background job を起動する方法は次の通りである.

command &

すなわち, 通常の命令列の最後尾に&を付加すれば良い.

さらに, 今, foreground で実行されている job を background で実行させたい時には, C-z を押すと, 現在の foreground job が suspend されるから, そこで

bg %*job-number*

という命令を実行すれば良い. ここで, *job-number* は,

```
jobs
```

というコマンドで調べることができる. 各行の最後の欄が job 名であり, 先頭に出てきた数字がそれに対応する *job-number* である.

2.6.6　.cshrc

.cshrc というファイルは C–shell が起動された時に実行される. ここに C–shell が立ち上がった時に実行して欲しい設定を書いておけば良い.

すなわち, path と幾つかの環境変数, 及び shell 変数を設定する. また, coredumpsize などの制限もここに書く.

C–shell は対話的に使われるとは限らないので, 対話的ではない時には設定してはいけないものは設定はしないようにするべきである.

2.6.7　.login

.cshrc が C–shell が立ち上がるたびに実行されるのに対して, .login は login の時だけ実行される. C–shell がもし login shell だった場合.cshrc が実行された後に実行される.

2.6.8　tcsh

tcsh というのは Shell の一種である. そしてその名のとおり, *csh* に幾つかの機能を付け加えたものである. だから, csh の命令は完全に実行可能である. version 5 までの tcsh は csh に対する差分のファイルしか配られていなかった. よって, tcsh を正式に使えるのは Unix のソースコードライセンスを持っているマシンだけであった.

tcsh は C-shell に主にファイル名補完機能とコマンドライン編集機能とを付け加えたものである. 他にもいろいろ機能があるのだが, 解説するのはここではこの 2 つにとどめておく.

[14] see *jobs*, *fg*, *bg* on manual page

き込み, 実行の許可を示している. ディレクトリの場合は, 実行許可はディレクトリ内のサーチを許可する. 許可は次の通りである.

- r ファイルが読み取り可
- w ファイルが書き込み可
- x ファイルが実行可
- - 指定された許可は認められない

このモードを変更する命令が chmod である. chmod は一般に

chmod *who*op*permission*

という形式である.

who は u, g, o, a が許され, それぞれ, user, group, others, all に対しての許可になる.

op は+, -, =である. それぞれ追加, 削除, 代入を表す.

permission は r, w, x 等で, それぞれ read, write, execute を示す.

これらの組み合わせでモードを変更する. すなわち, ファイル foo に他人に対しての読みとり許可を追加したければ,

```
chmod o+r foo
```

とするわけである.

2.6.3 ハッシュ

あるプログラム A は path に入っている. 実行許可も出した. でも実行されないということがある. これはまだ shell の実行許可ファイルを呼び出す機構に登録されていないからである. shell は命令が入力される毎に path に従ってファイルを探すのではない. そのかわり, path に登録された実行ファイルを高速に呼び出す表を作り, それを引いている. この表の表現方法に hash という技法を使っている.

この表に新たに項目を付け加えたい時には, rehash という命令を使う. これによって現在の path にある実行ファイルが全てこの表に登録される.

2.6.4 変数

何らかの文字列を覚えさせておいて, 後々使いたい時は, その文字列を shell 変数に入れておけば良い. C–shell の場合, set とすれば shell 変数を見ることができる. 変数への代入は

set *foo*=asdqwe

というようにする.

shell が呼び出したプログラムに値を渡したい時がある. プログラムは環境変数の値によって色々な動作を決定するようにできているものがある. 例えば, man を読む時にどの pager を使うかなどが個人毎に指定できるわけである. そのような時に環境変数が使われる. 環境変数は env で見ることができる. 環境変数への代入は

setenv *FOO* asdqwe

のようにする.

2.6.1　Shell 変数 path

さて, プロンプトが出ているところで, 我々は命令を打ち込めば, その命令が実行されるわけだが, その仕組みを考えよう.

例えば, 我々は今いるディレクトリのファイル一覧を見るために *ls* という文字列を入力する. この文字列を解釈し, 実際に ls という動作をするプログラムを起動しなければならない. しかし, ls というプログラムは何処にあるのだろう?実をいえば ls というプログラムは/bin/ls という名で管理されている. どのようにしてこのプログラムと *ls* という文字列を結びつけるのだろうか. それは変数 path である[12]. path の中に保存されている値を見てみよう.

```
geoph:/home2/s22516> echo $path
```

と入力すると

```
/home2/s22516/bin /usr/local/bin /usr/X11R5/bin /usr/lang
/usr/ucb /bin /usr/bin /usr/etc /usr/games /usr/local/bin/mh
/usr/local/bin/Wnn4 .
```

のようになる. この中に/bin という項目がある. この項目によって, /bin にある実行可能なプログラムはその名前を打つだけで実行できるようになる.

気をつけるべきことは, /bin の下ではなく, /bin に実行可能ファイルがあるということである. 例えば/bin/foo と/bin/bar/baz という実行可能ファイルがあり, path に/bin はあるが, /bin/bar はないとする. この時, /bin/foo は *foo* と打つだけで実行できるが, /bin/bar/baz は *baz* と打っただけでは実行できない.

逆に言うと, あるプログラムが呼び出せない時, まず, そのプログラムがある場所を path が含んでいるかどうかをまず疑うべきである.

また, path があるディレクトリ A を含んでいる時, 「A に path が通っている」という表現をとることもある.

2.6.2　ファイルのモード

Unix は複数の人間で計算機を使うことを前提とした OS であるから, 他人にファイルを消されたり, もしくは, 秘密のファイルを読まれたりしないような仕組みがある.

何かファイルがある場所で,

```
ls -l
```

としてみよう. すると,

```
drwxr-xr-x  3 s27082       1536 Apr 22 20:13 foo
-rw-r--r--  1 s27082     139573 Apr 12 14:20 bar.ps
```

というような表示が出てくる. ここで問題にしたいのは, 各行の先頭の 10 文字である. ここを Permissions Field と呼ぶ.

最初の文字は d, b, c, l, p, s, -のいずれかである. だが, 通常は d か-を見ることになるだろう. これは, d ならディレクトリ, -なら通常のファイルを意味する.

次の 9 文字は 3 文字ずつの 3 組に分けられる. 各組は所有者, 同一のグループ[13] に属する他の利用者, その他の利用者への許可を示している. 各組の中では, 3 文字がそれぞれ, ファイルの読みとり, ファイルの書

[12] MS–DOS を使用している者にはお馴染みであろう
[13] 現在の設定では, geoph のユーザは皆同一のグループ guest に属している.

```
lprm <ジョブ番号>
```

です. ここで, ジョブ番号は lpq で調べます. lpq の出力で Job と書かれた欄にある番号がジョブ番号です. もう一度, lpq を行って, きちんとキャンセルされたことを確かめましょう. lprm は, 印刷を行なおうとした端末でしかできません. どこで, 印刷したかわからないときは, 適当な端末で, lprm としてみればわかります.

日本語を含まないファイルで, 様々なオプションをつけたファイルの出力を行うには, enscript を使います. man enscript で, マニュアルが出ますので, これで, 理解してください.

例として次のようなものはどうでしょう.

```
enscript -2r -fTimes-Roman8 -FHelvetica8 <filename>
```

日本語を含むファイルを出力する場合には, 漢字コードに注意する必要があります[11] 端末室のプリンターは, euc 対応なので, それ以外の漢字コードのファイルの場合, 出力する前にコードを変換しておかなければなりません. euc 以外の漢字コードを含むファイルの名前が foo だとすると, その漢字コードを euc にするためには, nkf というコマンドを使い,

```
nkf -e foo > temp
mv temp foo
```

のようにします.

emacs で作ったファイルの漢字コードは, (地物標準環境の) 標準では euc になっています. 詳しくは, 2.9 節を見てください.

euc コードなら lpr で出力することができます.

日本語を含むファイルを印刷するもう一つの方法は, jtops というコマンドを使うことです. これは, テキストファイルを PostScript ファイルに変換するフィルターで, 例えば,

```
jtops <filename> | lpr
```

とすれば, print <filename> と同じことができます. こちらは, たくさんのオプションを設定することができ, かつ, PostScript ファイルが出力されるので, いろいろと書き直すことができます. PostScript とは, 様々な印刷を可能にする印刷機のための言語です. PostScript については 5.8 節で解説します.

その他, C などのプログラムリストをきれいに出力する vgrind や TEX もあります. このマニュアルも jLATEX で書いあります.

TEX の説明は 6.1 節でします.

紙はプリンターの左にある箱に入っています. 使用済みと新品とに分けてありますので, 適宜使い分けてください. テストなどには使用済みのものを使ってください.

2.6 shell

Shell は利用者が OS を利用する為の窓口である. 基本的には利用者が入力した文字列を解釈し, その文字列に対応するプログラムを起動するものである.

計算機を立ちあげるとプロンプトが出ている. 例えば geoph:/home2/s22516> のようなものである. これは Shell が出したもので, Shell が入力を受け付けていることを示している.

[11] 広く使われている漢字コードの体系は, euc, shift-JIS, JIS の 3 種類です. UNIX 上の多くのソフト (例えば, 日本語 emacs) は, euc を基本としています. また, MS–DOS のパソコンは shift-JIS を基本としています.

A プリンターのジョブは login 名と lpr を実行したマシン名で区別されます. lprm を実行するマシンは lpr を実行したマシンでなければなりません. (2.5 節参照.)

2.4.6　良く出てくる “foo” って何ですか?

Q 色々な本の説明を読んでいると “foo” という文字を良く見かけるのですが, これは一体何ですか?

A foobar に対しては次の説明が一番詳しいでしょう.

> **FOO** 1. [from Yiddish "feh" or the Anglo-Saxon "fooey!"] interj. Term of disgust. 2. [from FUBAR (Fucked Up Beyond All Recognition), from WWII, often seen as FOOBAR] Name used for temporary programs, or samples of three-letter names. Other similar words are BAR, BAZ (Stanford corruption of BAR), and rarely RAG. These have been used in Pogo as well. 3. Used very generally as a sample name for absolutely anything. The old 'Smokey Stover' comic strips often included the word FOO, in particular on license plates of cars. MOBY FOO: See MOBY.
>
> — *'Jargon file'*

要するに, どんな名前でも良い時に良く使います. 類義語として bar, baz, asd, qwe, poi, hogehoge, ika, tako などがあります.

2.4.7　path に入っているのに実行できない

Q path に入っている実行ファイルを実行できない.

A 実行モードはチェックしましたか? rehash はかけましたか? (2.6.2 節, 2.6.3 節参照.)

2.5　教育用計算機センターでのプリンターの使い方

本節では教育用計算機センターでのプリンターの使い方を説明します. 分散配置 Sun に接続している富士通日本語 PostScript 対応のプリンターを使う場合, 若干知っておかなければならないことがあります. 基本的には地球惑星物理学科でも同じなのですが少し違うところもあります. どこが違うのかは実際にやってみて調べて下さい.

まず, 紙の無駄を防ぐために, 表紙を出力しないのをディフォールトとします. このためには, .cshrc (2.6.6 節参照) に,

```
setenv ENSCRIPT -h
```

という 1 行を追加します.

ファイルの出力は, 標準では,

```
lpr <filename>
```

です.

プリンターに送られているファイルを調べるコマンドは,

```
lpq
```

です. ちなみに, lpq の q は, 待ち行列 (queue) の q です.

プリンターを詰まらせると後の人が困りますので, 印刷を行ったはずなのに, 印刷されなかったときは, まず, lpq でプリンターの状況を調べ, もしまだプリンターの待ち行列に自分のファイルが詰まっていたら, 出力をキャンセルします.

出力をキャンセルするコマンドは,

2.4.1 core

Q 知らない間に私のディレクトリに `core` というファイルができている. これはどういうファイルなのですか. 誰かにここに書き込まれたのですか.

A `core` というのは実行しているプログラムが何らかの原因で死んだ時に, それが使っていたメモリの様子をファイルにして残しておいてくれるものです. これは dbx というソフトで解析できます. しかし, 一般に非常に大きなファイルが出来てしまいますから, かなり「良く分かった人」でないと解析は難しいでしょう. また, 非常に大きなファイルであることが多いですから, いらない `core` は消しましょう. `core` を吐かせたくない場合は `.cshrc` (2.6.6 節参照) に `limit coredumpsize 0` と書いておきましょう.

2.4.2 #FOOBAR#

Q 知らない間に私のディレクトリに `#foobar#` というファイルができている. (`foobar` は例です.) これはどういうファイルなのですか. 誰かにここに書き込まれたのですか.

A `foobar` というファイルに見覚えありませんか? `#foobar#` というファイルは emacs が作ったファイルです. マシン及び emacs が何らかのアクシデントで落ちるかも知れないので, ある一定のキー入力数毎に現在のバッファの内容を `#foobar#` というファイルに自動的にセーヴしています. これを auto-save と言います. emacs を使っている時に時々エコー領域に "Auto-save" とメッセージが出ることがありますが, それはこれをしているのです.

Auto-save ファイルはファイルをセーヴ命令でセーヴした時に無くなるはずです. もし, Auto-save ファイルが残っているのであるならば, それはそのファイルはセーヴされずに emacs が終了させられたということです. もしかしたら大事な編集結果が失われているかも知れません. emacs を終了する前には必ず必要なファイルはセーヴするように. また, X-window を終了する時には emacs を確実に終了させてから終了するようにしてください.

2.4.3 foobar~

Q 知らない間に私のディレクトリに `foobar~` というファイルができている. これはどういうファイルなのですか. 誰かにここに書き込まれたのですか.

A `foobar~` というファイルは emacs の残す back-up のファイルです. (`setq make-backup-files nil`) という 1 行をホームディレクトリの `.emacs` というファイルに書いておけば, emacs は back-up ファイルを残さなくなります.

2.4.4 ファイルが消せない.

Q 頭に- という文字を含んだファイルを作ってしまい消せない.

A 例えば`-foobar` というファイルを作った時には, `rm ./-foobar` という様に, 頭に`./`をつけてやれば消せます.

2.4.5 プリンターのキューが消えない

Q lprm しても Permission denied といわれてキューから外せない.

大物の殺し屋

それでも消えず, かつ, どうしてもそのプロセスを消したいときは, 最終的な手段を講じます.

```
kill -KILL <おかしくなったプロセス識別子>
```

とします. すなわち,

```
kill -KILL 300
```

とします.

kill をかけるときにはこの順序で行なって下さい. というのは, ただの kill の時には, プロセスが死ぬ準備をしてから死んでくれます. ところが, kill -KILL の時にはすぐに殺すので, プロセスが死ぬ前にしなければならないことをやらずに死んでしまいます. 特に X:0 というプロセスにはやらない方が良いでしょう.

ps と入力しても, おかしくなったと考えられるプロセスがない場合は, ps ux として, もっと詳しいプロセスの状況を知ることができます.

2.3.5 Destroy Window

以上のことを行ってもどうしてももとに戻らないときは, そのウィンドウを消します. それには Destroy Window を使います. 標準環境での Destroy window は, root window で, 左のボタンを押し, Kill をボタンを押しながら選びます. すると, しばらくして, マウスのカーソルがドクロマークに変わりますから, 殺したい window にマウスを持っていき, 左ボタンを押すと, window を kill することができます.

2.3.6 他の計算機から入る

ウィンドウの中のコマンドがうまくいかなかった場合は, 以上で, 決着がつくはずです. しかし, X-window 全体がおかしくなって, その端末では入力すらできなくなることがあります. 例えば, X-window から, Sunview を立ちあげたりするとこうなります. このときは, 別の端末から rlogin でその端末に入ります. つまり,

```
% rlogin <おかしくなった端末の番号>
```

とし, 上と同じように, ps と kill を行います. 例えば,

```
% rlogin geoph
```

とします. また, おかしくなった端末のキー入力が変になっていたら,

```
% kbd_mode -a
```

とします. これでもどうしようもない場合は, リブートすることになりますが, これは初心者はやらないほうが良いと思います. 指導員かまわりの詳しそうな人に頼んでやってもらいましょう.

2.4 FAQ (Frequently Asked Questions)

良くある質問のことを業界では FAQ と呼びます. 本節ではそういった質問とそれに対する答とをいくつかまとめてみました.

2.3 トラブルに落ちた時

Unix はあまり初心者にとって使い易くは設計されていません. 様々なヘルプ機能もそれをうまく呼び出す術を心得ていなければ使えません.

当然, うまく動かなくなることがよくあります. このようなとき, いきなり電源を切ってはいけません. 機械が壊れてしまいます.

では, どうしたら良いのでしょうか.

2.3.1 キーボードからの中止の命令

まずは, C-c と打ってみましょう. (C-c とは, Ctrl キーをおしたまま c というキーを押すことです.)

2.3.2 画面出力の再開

次に考えることは, 間違って C-s を押してしまったのではないかということです. C-s を押すと文字の出力を行なわなくなってしまうのです. C-q を押すと文字の出力を再開する機能が働きます. もし C-s を押していたのなら, これで解決します.

2.3.3 よくある, 終了の儀式

それでもだめなら, 何か間違ったプログラムが動いている可能性があるので, q とか, quit とか, C-d とか, exit などと入力してみましょう. 例えば, cd というコマンドを誤って, dc としてしまった場合は, q とすればもとに戻ります.

2.3.4 殺し屋を雇う

それでもなおだめならば, 別のウィンドウ (kterm) を開いて, そこで, ps という命令を打ちます. ps とは, プロセスの状態を見るコマンドです. プロセスとは, 今, 計算機が行っている仕事だと思ってください. 話をもとに戻して, ps と入力すると出てくる画面で, 一番左の数字をプロセス識別子といいます. この情報を見ながら, おかしくなったと思われるプロセスを強制的に消滅させます. その命令を kill といいます.

小物の殺し屋

```
kill <おかしくなったプロセス識別子>
```

とします. 例えば,

```
kill 300
```

とするわけです. もう一度, ps と入力して見れば, 本当にプロセスが消滅したかがわかります.

中物の殺し屋

まだ殺せない時には, ちょっと手段をエスカレートさせます.

```
kill -HUP <おかしくなったプロセス識別子>
```

とします. すなわち,

```
kill -HUP 300
```

とします.

ソースを確認する

今セーブしたばかりなので特に必要はないとは思いますが, ls と less をつかって今作った hello.c が存在することと, それが正しく出来ていることを確認してください.

コンパイルする

それではコンパイルします. geoph に標準でついている C コンパイラは, 古い K&R スタイルしか受け付けないので, ANSI のスタイルを使えるコンパイラを使います. コマンド名は gcc です.

```
geoph:/home2/s22516> gcc -o hello hello.c
```

これでコンパイルできます. -o hello というのはコンパイル結果を hello という名前にするように指定するものです. (何も指定しないと a.out という名前になります.) エラーがあるときは, マウスを emacs の方に動かして, エラー箇所を修正し, セーブしなおしてから, 再コンパイルします. 特にエラーがないようでしたら, ls でちゃんと hello が出来たことを確認してください.

hello を実行する

それでは今出来た hello を実行してみます.

```
geoph:/home2/s22516> hello
```

これでちゃんと 'hello world.' と表示されれば成功です. 後は UNIX の問題というよりもは, C の問題です.

グラフィックスについてひとこと

ひととおり C でプログラムを作れるようになると, グラフを出したくなる人が多いようです. すると 「グラフィックを使うためのライブラリはないのですか ?」という質問をよく聞くようになります. 結論からいうとあります. しかし本当に必要なのでしょうか ?UNIX には tool の精神があって, それぞれのプログラムは出来るだけ単機能に作っておき, それらを組み合せて大きな機能を実現するという思想があります.

私がある計算をするプログラムをかいて, その結果をグラフにしたいときは, プロットしたい点の座標を 1 行につき 1 点ずつ x　yの順に出力するようなプログラムをかきます. つまりこんな感じです.

```
10 50
20 56
30 59
40 66
```

そして, それを xgraph や gnuplot[10] といったプログラムに渡して, グラフにさせます. この方がプログラムを作るのは楽になり, かつ出力結果は美しいものになります.

もちろん高度に対話的なプログラムを書きたい場合はグラフィックライブラリが必要になるでしょう. しかし多くの場合はそれだけの労力をかける必要はないのです. ぜひ皆さんも超統合型アプリケーション的発想のみではなく, ツールを使う UNIX の使い方に挑戦してみてください.

[10] これらの使い方は後の章で説明します.

用できます.

2.2.12 エディタ vi

emacs と違って, vi は UNIX には必ず付いているエディタです. geoph には emacs がインストールされていますが, どの UNIX にも emacs がある訳ではありません. 従って, vi というエディタがあるということぐらいは覚えておいて下さい.

vi の具体的な使い方は解説しません.

vi は非常に癖のあるエディタで一般には余り好まれていないようですが, ESC と DEL 以外の特殊キーを使う必要がないので, ブラインドタッチ[8] で使うには便利であるという長所もあります[9].

2.2.13 C で簡単なプログラムを作る

さてそろそろ実践編に入ります. 実践編といっても非常に簡単なこと一つやってみるだけですが...

これから先の説明はここまでのことについて分かったものとして行います. 多少不親切かも知れませんが, 読み返しながらやってみてください.

ソースファイルを作る

作るプログラムは C 言語のバイブルにもあるように 'Hello world.' と表示するだけのものです. 文法は ANSI C に従うものとします.

ソースファイルは当然 emacs をつかって作ります. emacs で `C-x C-f hello.c↓` として, `hello.c` を作ることを指示してください. そしてそこに,

```
#include <stdio.h>

void main(void){
  printf("Hello world.\n");
}
```

とかいてください. インデントは今はタブキーを使わずにスペースキーを使ってください. (このわけについては, emacs の C-Mode について知る必要があります.) `C-x C-s` でセーブしてください.

printf のマニュアルを見る

今はすんなりとソースが書けましたが, いつもこうとは限りません. 仮に `printf` 関数の使い方がよく分からなくなったとしましょう. オンラインレファレンスで調べることができます.

マウスカーソルを kterm の方に移してください. プロンプトに対して

```
geoph:/home2/s22516> man printf
```

と入力してください. `printf` の詳しい説明が読めます. このように, `man` コマンドは, コマンドのみならず関数の説明も読むことができるのです. 関数とコマンドで同名のものがあるときの対処の仕方は `man man` に書かれています.

[8] 手元を見ないでキーボードを叩くこと.

[9] emacs は ESC や DEL 以外にコントロールキーを使います. コントロールキーの位置や大きさはメーカーによって違うので, どのメーカーのキーボードでも同じようにブラインドタッチで使えるという訳にはいきません.

tutorial を終了させて次に進みたい場合です. 本当は tutorial のことを気にせずにそのまま次の作業に移ればいいだけなのですが, 私がそうであったように何か精神的に落ち着かない人もいることと思います. そういう人は, C-x k↓ と入力すれば tutorial の画面が消えます.

info

emacs の詳細な使い方は info を使うことによって知ることができます. info とは emacs 内での man の様なものです. info 自身の使い方も info で見られます. ただし, info で得られる情報は膨大です. 初心者にはあまりお勧めできません. しかし, やがては全てを知りたくなるでしょう. そのときには 「emacs の全ての説明は emacs の中にある」ということを覚えておいてください.

> emacs 中で, M-x info↓ と入力してください. 画面の説明の通りですといえばおしまいなのですが, それではあまりにも不親切なので, ここでは h(これを押すと何が起こるか画面の説明を読んでから押してください) を入力するといいでしょう. これで, info の使い方が覚えられますから, 後は emacs の説明を読めるようになるでしょう.

最低限の操作

emacs にはエディターとしての機能とそれ以外の機能がありますが, ここではエディターとしての機能のみ解説します. 表記などについては 2.2.1 節も参照して下さい.

emacs を立ち上げた直後に C-x C-f として編集するファイル名を指定して下さい. アルファベットなどの普通の文字は単にキーから入力するだけで画面に書き込まれます. 編集を終了する場合には C-x C-c を入力して下さい.

以下に, 基本的なキー操作を挙げます. “ミニバッファ” というのは emacs の画面の一番下の部分のことです.

終了	C-x C-c	
エラーからの脱出	C-g	(エラーからの脱出 · コマンドの実行中止)
編集するファイルの指定	C-x C-f	(ミニバッファにファイル名を入力)
ファイルへのセーブ	C-x C-s	(そのままの名前でセーブ)
	C-x C-w	(別の名前でセーブ.　ファイル名はミニバッファで指定)
カーソルの移動	C-p	上
	C-n	下
	C-b	左
	C-f	右
	C-a	行の先頭
	C-e	行の最後
削除	C-d	カーソル位置の 1 文字
	C-k	カーソル位置から行末まで
	C-h	カーソル前の 1 文字
	<*Back Space*>	同上
	C-y	C-k で削除された文字列の復活
操作の取消	C-x u	
ヘルプ	<*Del*>	

ただし, 標準の emacs ではカーソル前の 1 文字の削除は<*Del*> に割り当てられていますが, 現在の環境ではこのように C-h と <*Back Space*> とに割り当て直しています. また, カーソルの移動にはテンキーも使

```
Yoshida Shigeo, (room 417 ext 4296 home 03-3991-8464)
```

の様に名前と部屋番号, 内線, 自宅の電話番号をいれておきましょう. 使えるのはアルファベットと数字だけです, またカンマは名前とそれ以降の間に一つだけいれてください.

本名を設定しておくことの御利益を説明しておきます. 一つは finger というコマンドによってユーザー ID から本名を, または本名からユーザー ID を知ることができます. 例えば, finger yoshida@geoph とやってみてください. しばらく待っていると指定したユーザーの ID, 本名などの情報が表示されることでしょう.[6] finger shigeo@geoph でも同様の結果が得られます. 詳しくは man finger してください. なお, コマンドの後半の@geoph はなくても構いませんが, あった方が応答が速いでしょう. (理由はここでは説明しません.)

他にも, この冊子では使い方を説明していませんが, 電子メールを出したり, ネットワークニュースに投稿する際に, 差出人の欄にここで登録した名前が入ります. (登録していないと s22516 の様に ID がそのままはいってしまいます.)

2.2.11 エディタ emacs

emacs の起動

さて, 次に必要なものはエディターです. エディターは kterm とは別に新しいウインドウを作らせて実行させます. ここでは emacs というエディターを使います. シェルのプロンプトに対して,

```
geoph:/home2/s22516> emacs &
```

と入力してください. しばらくして画面右上に emacs が開きます. 地物標準環境では X-window 起動時に自動的にこのコマンドが実行されるようになっています.

このエディターは非常に多機能で強力なのですが, MS–DOS のエディターに慣れている人にはかなりのくせものです. 一日でも早く便利だと思えるようになってくれると, このエディタのファンとしては嬉しいのですが, まずは簡単な使い方から覚えてもらうことになります.

emacs の終了

emacs の終了は C-x C-c で行います. セーブしてないファイルがある場合にはそれらをセーブするかどうか順に聞いてきます. y または n で答えてください. ここで一つでもセーブしなかったファイルがある場合, セーブしてないファイルがあるけど本当に終了してよいかどうかを聞いてくるので, 終了してもよい場合は, yes↓と答えてください. no↓ と答えると終了せずに編集モードに戻ります.

tutorial

最低限使えるためのキー操作などは後の方にまとめて書いておきますが, emacs の使い方を学ぶには, emacs の tutorial (個人指導) を使うことから始めると良いでしょう. 呼びだし方は, (マウスが emacs のウインドウに入っているのを確認した上で) M-x help-with-tutorial-for-nemacs↓と入力してください[7]. 後は画面の指示に従ってください.

二つほど補足しておきます. 一つは, tutorial には, emacs を一時的に終了させるために C-z を使うように書いてあると思いますが, X-Window 上で emacs を使っている以上 C-z は使う必要はありません. emacs は一回起動したら, ログアウトするまで終了させないでそのままにしておきましょう. もちろん, 使っていて分からなくなった時に, C-x C-c で終了させて, またやり直すのはなんら問題はありません. もう一つは,

[6] ECC(教育用計算機センター) には大勢の人が登録されていますから, かなり応答時間がかかるかも知れません.
[7] 最初のは表記表通りの記号で残りはそのまま入力してください.

less — ページャー

MS–DOS には MORE というページャーがあります. UNIX にも more コマンドはあります. less はその改良版のページャーです. 使い方は,

```
geoph:/home2/s22516> less foo
```

で最初の一画面分が表示されるので, (SPC) で一画面進み, b で一画面戻ります. 終了するときは, q です.

mkdir — ディレクトリの作成

新しいディレクトリを作ります.

```
geoph:/home2/s22516> mkdir ncsim
```

で ncsim というディレクトリを作ります. なお, MS–DOS と異なり, UNIX ではディレクトリの区切りを '/' で表します. またドライブという概念はありません.

cd — カレントディレクトリの変更

カレントディレクトリを変更します. ただし,

```
geoph:/home2/s22516> cd
```

のように何もディレクトリを指定しないと自分のホームディレクトリ[5]に移ります.

rmdir — ディレクトリの削除

から っぽのディレクトリは, この rmdir コマンドで削除することができます.

man — オンラインレファレンス

ここであげたコマンドを初めとする多くのコマンドのマニュアルは計算機を使いながらその場で見ることができます. そのためのコマンドが man コマンドです.

```
geoph:/home2/s22516> man man
```

としてみましょう. man コマンドの説明が見られます. man コマンドは自動的に less コマンドを起動しますから, 進んだり戻ったりする方法については前述の less の項を参照してください. ここであげたコマンドは全てマニュアルがありますから, 詳しい説明はこちらを使って覚えてください. 慣れないうちは (いえ, UNIX に慣れ親しんでからでも), 2 つ kterm を開いておいて, 片一方にマニュアルを表示させておいて, もう片方でコマンドを実行するといった使い方をすればよいでしょう.

chfn — 本名の登録

これは必須事項です. 本名が登録されていない場合は必ずこのコマンドを実行して登録して下さい.

```
geoph:/home2/s22516> chfn
```

としてください. 説明が数行出てから, 名前の入力を促されるので,

[5] UNIX はマルチユーザーシステムですからお互いの邪魔をしないように各ユーザーに付き一つのディレクトリ (ホームディレクトリ) が割り当てられます. 通常は皆, 自分のディレクトリより下で作業し, 共有ファイル (例えば皆が使うプログラムなど) のみを共有の場所においておきます. ログイン直後には各ユーザーは自分のホームディレクトリがカレントディレクトリになっています.

2.2.10 shell からのコマンド

いままで度々プロンプトといってきましたが, このプロンプトを表示しているプログラムがシェル (shell) です. シェルには何種類かあって, sh (シェル), csh (C-shell), tcsh といった名前が付いています. 我々が通常使うのは, csh と tcsh です. geoph では デフォルトは tcsh になっています. どちらも初心者には同じ様に見えるはずなので特に気にする必要はないでしょう. なお MS–DOS ではシェルに当るものはコマンドインタプリタと呼ばれ, COMMAND.COM という名前が付いています. これらのプログラムの役割は人が入力した文字列 (コマンド) を理解してプログラムを起動することです. (cf. エディタは入力した文字をファイルに書き込むのが仕事) ではここから実行できる多種多様なコマンドの中からいくつか紹介してみることにします. なお, 以下の説明でもプロンプトは `geoph:/home2/s22516>` です. キー入力する時はこの `geoph:/home2/s22516>` は入れません. プロンプトに対して入力するということを表しているだけです.

MS–DOS ユーザーに注意を一つ. MS–DOS と UNIX ではワイルドカードの扱い方が異なります. UNIX になれるまではワイルドカードを使わないことを強くお勧めします.

ls — ファイルの一覧

MS–DOS でいうところの `DIR` コマンドです. `ls -l` とすると作成日時やサイズなどの情報も表示される様になります.

cat — ファイルの中身の表示

MS–DOS で言うところの `TYPE` コマンドです. `foo` というファイルを見たいときには,

```
geoph:/home2/s22516> cat foo
```

とします. 長いファイルを見る場合は, 後述の `less` を使うといいでしょう.

cp — ファイルの複写

`COPY` にあたるコマンドです. 但し, 必ず

```
geoph:/home2/s22516> cp foo bar
```

の様にコピー先ファイル名 (またはコピー先ディレクトリ名) も指定しなければなりません.

mv — ファイルの移動 / 名前変更

単純には DOS の `REN` コマンドのように, ファイル名の変更に使えます. それ以外にもディレクトリ間のファイルの移動にも利用できます. 後者の例をあげます.

```
geoph:/home2/s22516> mv foo old/foo
```

ここで `old` はすでに存在するディレクトリの名前です.

rm — ファイルの削除

`DEL` に対応するコマンドです.

```
geoph:/home2/s22516> rm foo
```

で `foo` というファイルが削除されます. MS–DOS の場合と違って, 誤って削除したファイルを取り戻してくれるようなユーティリティーはありませんから, 削除する場合は細心の注意を払ってください.

```
C-x C-c
```

と入力して下さい. その他の, 時計, 郵便受け, twm icon manager のウインドウについてはいちいち閉じなくても結構です.

> もし画面に何もないはずなのに一覧表 (twm icon manager) に他のものが出ている場合は (例えば xco と出ているとします), xco と書いてあるところのすぐ左にある四角に囲まれた x というところにマウスカーソルを持っていって, 左クリックします[4]. するといままで隠れていたウインドウ (ここでは xco) がでてきますから, それを終了させてください. 一覧表に x のマークが付いているということは, 現在それが表示されていないという意味です.

X が終了したら, ログインしたばかりのときのような大きな文字でプロンプトがでることと思います. ここまできたらたいてい次のコマンドは logout でしょう.

2.2.9　kterm

kterm の起動

kterm は, X 用のターミナルエミュレータ xterm の漢字が使えるバージョンです. といっても何のことだかピンとこないでしょう. 要は, kterm ひとつが X を起動する前の画面一つ分だということです. UNIX はマルチタスクシステムですから, 複数の窓から複数のコマンドを実行した場合それらは同時に実行されます.

MS–DOS との対比の例をあげます. あるプログラムを実行中にファイル名を聞かれたので DIR コマンドを使ってファイルの一覧を見たくなったとします. DOS では, いったんプログラムを終了するか, チャイルドプロセスで COMMAND.COM を呼び出して DIR コマンドを実行することになります. 豪快な方法としては, もう一台パソコンを使ってそいつで, DIR コマンドを実行する手もあります. このようなことを X で行うときは, kterm を二つ開いておき (起動しておき), 片一方でプログラムを実行して, もう一方で ls (UNIX での DIR みたいなものです.) を実行すればよい. ということになります.

さて起動の仕方ですが, 地物標準環境の設定では, X の起動時に画面左側に大きなウインドウが開いていることでしょう. それが kterm です. ですからすでに起動している人はそれを使えばよいでしょう. X 起動時に kterm が出てこない場合や, もっといっぱい kterm が必要な場合は, 次のようにして呼び出してください.

まずマウスを背景のグレーのところ (root window) に移動して中ボタンを押えてください. メニューが出てきますから, ボタンを押えたまま Kanji Terminal と書かれたところまでマウスを移動してください. 字の色が反転表示されたところで, ボタンから指を離します. これで呼びだし作業が終りました.

しばらくすると, 画面左上に geoph と書かれた小さな長方形が出てきて, マウスカーソルの位置に四角い枠が出てきます. マウスを移動してこの枠を見やすいところ (他のウインドウと重ならないところなど) に移動して左クリックするとそこに kterm のウインドウが開きます. もう少し待っているとプロンプトが出ることでしょう.

kterm の終了

kterm の終了は, プロンプトの出ているところで,

```
geoph:/home2/s22516> exit
```

と入力します. `There are stopped jobs.` と出たときには, 初心者はもう一度 exit と入力すればよいでしょう. これでウインドウが消えてくれるはずです. もしまだプロンプトが出るようでしたら, ウインドウが消えるまで exit を繰り返せばよいでしょう.

[4] マウスの左のボタンを押してすぐ離すこと.

2.2.6 地物標準環境のインストール

初めて geoph を使う場合には、地物標準環境をインストールします. プロンプトからインストールのコマンド,

```
geoph:/home2/s22516> /home2/aic/etc/setup
```

を実行します. これで, 必要な設定がなされるはずです。ただし、これが有効になるのは次回ログインしたときからなので、いったんログアウトしてから再ログインしてください. なお, 標準環境は geoph の設定の変更などに対応して順次変更されていますので、「ある日ログインしてみたらうまく動かなかった」などというときには、再度インストールしなおしてみるのもよいでしょう。

> 地物標準環境のインストールコマンドは、実際にはいくつかのファイルを自分のホームディレクトリーにコピーしてくるコマンドです。`ls -a` として表示される. (ピリオド) で始まるファイルのほとんどが標準環境のファイルです。これを自分で書き換えた人は、再インストールすることによって書き変わってしまうので注意してください。

2.2.7 X-Window の起動

次に X-Window (以下単に X と書きます) を起動します. X を使うとマルチウインドウ環境になるので UNIX の持つマルチタスク機能の力を存分に発揮させられます. またワークステーションで漢字を表示させるためには, X を使う必要があります.

X-window の起動のためのコマンドは,

```
geoph:/home2/s22516> xinit
```

です. 地物標準環境では login 時に自動的にこのコマンドが実行されるようになっています. このワークステーションでは, 多くのものを多くの人と共有していますから, 時と場合によって X-window が起ちあがるまでの時間やキー入力に対する応答時間が随分違います. が, キー入力に反応しなくなった場合でも 2 分ぐらい待てば何事もなかったように復活することがよくあります.

さて, 画面がグレーになって, 数個のウインドウが開くでしょう. ここからは, キーボードの他にマウスも使うことになります. また, これからはキーボードから入力した文字は, マウスカーソルの入っているウインドウに渡されます. つまりマウスがどこにも入っていない状態では, 何を入力しても全て捨てられてしまうということになります.

ここでひとつ注意事項です. **X の中から二重三重に X を起動しようとしないでください. また, sunview** というコマンドも使わないでください. ほぼ確実におかしくなるでしょう.

2.2.8 X-Window の終了

X システムを終了させる方法を説明します. 背景のグレーの部分 (これを root window と呼ぶ) にマウスカーソルを持っていって, 左ボタンを押します. そうすると Twm という表題のついたメニューが出てくるはずです. 左ボタンを押したまま, マウスを下げていって, 一番下の Exit の色が反転したところで, マウスを離します. これで, X システムが終了します.

上の方法だと, 終了させる時にいろいろと警告が出てくることがあります. それは, 今の方法は正しいお作法に従っていないからです. 正しいお作法は, 各種のウインドウを閉じてから, 上の操作をします. ウインドウの閉じ方は, 以下の通り. プロンプトが出ているウインドウについては, そのウインドウの中で

```
geoph:/home2/s22516> exit
```

と入力して下さい. emacs (2.2.11, 2.9 節) は, そのウインドウの中で

```
geoph login:  s22516
```

という具合です. すると次に

```
Password:
```

と聞いてくるので, これにパスワードを入力してください. このとき, 入力した文字は画面に表示されませんし, カーソルも移動しません. 正しく入力すると何やら数行メッセージが表示されて, 最後に

```
geoph:/home2/s22516>
```

などと表示されるでしょう. これをシェルプロンプト (shell prompt) または, 単にプロンプトと呼びます. これでログインしたことになります. パスワードが違う場合は, login: のところに戻りますから, もう一度やり直してください. 同様に, 入力を間違えたときは, ↓ を押して先に進めてしまえばやり直せることになります.

2.2.4　logout

ログインができたところで, 編集方針にしたがって, 端末使用終了 (logout) の方法を説明します. ログインしてから実行させたプログラムをそれぞれ所定の方法で終了させます. 特に, 地物標準環境をインストールした後は, X-window が自動的に立ち上がっているはずですから, X-window を 2.2.8 節, および 3.1.2 節に従って終了させます. そしてプロンプトが出ている状態から,

```
geoph:/home2/s22516> logout
```

と入力すると,

```
geoph login:
```

と終了し, 次の人のログイン待ちになるはずです. もしも There are stopped jobs. といわれてまたプロンプトが出るようでしたら, 取り敢えず初心者は, もう一度 logout と入力してください. これで終了できるはずです.

> 意図的に終了させないものは, 当然そのままで構いません. There are stopped jobs. と出る場合は, jobs と入力するとまだ終了していないジョブ (プロセス群) の一覧が出ます. running と表示されるものはバックグラウンドで走っているもの, suspended と表示されるものは実行停止中のもの, というように現在の状態が表示されています. いずれの場合も各行の先頭に [1] の様にジョブ番号が表示されていますから, kill %1 のように入力するとたいてい終了させることができます. 詳しくは後述の man コマンドで csh, kill について調べてみてください.

長時間端末を離れる時は logout するようにしましょう. トイレに立つ程度ならまだうなずけますが, 食事に行くなど長時間席を離れる時には, 端末を独占するのは考えものです. また, 人がついていなくてもできるような計算は (この冊子ではその方法は説明しませんが) バックグラウンドで実行させておいて[3], できるだけ端末は他の人に譲ってあげましょう. 他にもユーザーがたくさんいることを頭に入れておきましょう.

2.2.5　パスワードの重要性

パスワードは必ず設定しておかなければなりませんし, 他人に教えたり, ファイルに書き込んでおいたりしてはいけません. また, 推測できるような簡単なものであってはなりません.

我々のワークステーションは世界中とつながっています. パスワードが盗まれれば, 招かざるものが侵入してきてシステムに重大な損害を与えないとも限りません. ワークステーションを使っている時には, システムを危険に晒さないようにする義務を負っているのだと考えて下さい.

[3] あなたのコマンドを「裏側で」実行しながら, なおかつ, 他の人がその端末で仕事をすることが出来るようにすることが出来ます. このような実行の仕方を "バックグラウンドで実行する" と言います.

することを許しているがゆえに, すなわち, できることが非常に多いがゆえにとっつきにくくなっているのです.

2.2.1 表記・その他の約束ごと

キーボードから入力する場合, 特殊な文字については以下のように表すことにします.

C-x	Control キーを押えたまま x を押す.
M-x	Meta キー (スペースの両隣にある菱形印のキー) を押えたまま x を押す. または, ESC キーを押してから x を押す.
↓	改行キーを押す. 特に明示する必要がある場合のみ使います. 通常, 入力するように指示された文字が, 1 文字ならば改行キーは押しません. 複数文字ならば文字列の最後に改行キーを押します.
(SPC)	スペースキーを押す. 特に明示する必要がある場合にのみ使います.

また,

> このように小さな文字で書かれた部分は, 初めて読む人は飛ばして読んでも構わないと思われるものです. 基本的にこのガイドには詳しい説明を書かない方針ですが, 場合によってはこのような形で多少説明します.

このガイドの中で何かプログラムを起動する方法を説明する場合, その直後に終了させる方法を説明することにします. たいていの場合は, 起動した以上何か作業をするのでしょうが, 年度の初めには終了させられずに放置してある端末を度々見かけますから, このような順序で紹介することにします.

2.2.2 トラブルに陥ったとき

キーボードからの入力を何も受け付けなくなるようなことがたまに起こります. このような時どうしたら良いかについては, 2.3 節を参照して下さい. ここでは, いきなり電源を切ってはいけないことだけを強調しておきます.

2.2.3 login

UNIX はマルチタスク, マルチユーザーのシステムです. ログインとは, 誰がこれから計算機を使おうとしているのかを明らかにし, 他のユーザーと区別するための操作です. このおかげで, 例えば, 他人にみられたくないファイル (e.g. メール) を他人が読めないようにしたり, 他人に自分が作ったファイルを消されたりしないようにするなどの設定が可能になります.

ログインするためには, ユーザー ID とパスワードが必要になります. 地球惑星物理学科, および教育用計算機センター (以後 ECC, Educational Computer Center とします) におけるユーザー ID は, 理学部生の場合は学生証番号の前に小文字の s をつけたものになります. 教育用計算機センターで使うパスワードについては, 各自パスワード表をもらったことでしょう. UNIX では, 基本的に小文字を使います. パスワード表には, 大文字で書いてあることと思いますが, 小文字を使ってください. また, 地球惑星物理学科で使うパスワードはユーザー登録時に各自で申請したものを使います.

それでは, 地球惑星物理学科のサーバーである geoph にログインします.

```
geoph login:
```

などと画面に表示されている端末を探してください. 明らかに人が使っている端末は論外ですが, 画面が真っ暗になっている場合や, 画面いっぱいに絵が表示されている場合は, ↓を押してみると上のような表示が出てくることも多いです. そこに自分のユーザー ID を入力してください. 例えば,

第 2 章

UNIX ワークステーションの使い方

> The White Rabbit put on his spectacles. "Where shall I begin, please your Majesty?" he asked.
> "Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."
>
> Lewis Carroll *Alice in Wonderland*

2.1　地物標準環境

2.2 節以降に, "地物標準環境" とか "標準環境" という言葉が出てきます. これは, 東京大学理学部の地球惑星物理学科地球惑星物理学演習テキスト編集グループがワークステーション上での仕事をしやすくするために作成した「環境」です.

UNIX はユーザーの好みに合わせて様々な設定を変えること (カスタマイズといいます) の出来る柔軟な OS[1] ですから, 同じ UNIX でも設定 (「環境」) によって使い方が大きく違うこともあり得ます. もちろん, 設定の仕方を知っている人にとってはその違いは表面的なものですが, 初心者に UNIX の解説をする場合には, どの UNIX にも共通する部分の解説だけでは実用にはなりません. そこで, 2 章から 6 章では, 地物標準環境に即して解説をすることにします.

従って, まず 2.2.6 節に従って地物標準環境をインストールする必要があります.

また, 2 章から 6 章を読む前には, 7 章も一度は必ず読んで下さい.

2.2　とりあえず使えるために

この節は地球惑星物理学科の Sun ワークステーション "geoph"[2] を初めて使う人を対象に,「取り敢えず使える」ことを目的として書かれています. よって, さし当って必要がないと思われることに関しては正確さよりも単純さを重視して説明します. また全てを最初から説明するのは無理があるので, 教養学部での講義もあわせ考え, MS–DOS におけるファイル, 簡単なコマンド, ディレクトリなどについては知っているものとして話を進めます. また方針としては, 多くのことをここで説明することは避け, どのようにしたらより詳しい使い方を知ることができるかを説明していくことにしたいと思います.

初めての人には, このワークステーションは非常にとっつきにくいものに感じられるかも知れません. しかし, 多くの場合, それは便利さや自由さの裏返しなのです. このワークステーションで走るオペレーティンシステム UNIX, 及びその上で動作する多くのプログラムは, 使う人が好みに応じて大幅にカスタマイズ

[1] Operating System

[2] 学部学生用のマシン vsgold,vsiron も使い方はほぼ同じです。ですから geoph を vsgold あるいは vsiron と読み換えてしまっても問題ありません。

器機名	設置場所	IP アドレス	ニックネーム	担当
UT Net 基幹リング ルーター	大型計算機 センター	133.11.3.1	sgw3-e	大型計算機センターネットワーク管理掛
国際理学ネット ルーター P4200 (現在は使用されていない)	理学部 3 号館 026	133.11.3.2	uts3gw	TISN
大型計算機センター HITAC VOS3	大型計算機 センター	130.69.240.2	knet	大型計算機センターネットワーク管理掛
大型計算機センター DEC5830	大型計算機 センター	130.69.240.3	tansei	大型計算機センターネットワーク管理掛
教育用計算機センター S サーバー	教育用計算機 センター	130.69.242.2	ecc-s4	教育用計算機 センター
国際理学 LAN ネットワークサーバー SUN4-280/S	理学部 7 号館 情報科学科	133.11.11.11	utsun	TISN
地球惑星物理学科 ネットワークサーバー Sun SPARC station 2	理 3 号館 320	133.11.3.163 133.11.24.1	geoph	研究スタッフ室

サネットとの間にはやはりリピーターが必要となるが, さらに, 光と同軸との信号の変換のために光トランシーバーが必要になる.

ネットワーク同士を結ぶためには信号の出入りを制限する関所, ゲートウェイが必要となる. ゲートウェイとしては, UNIX マシンをもちいることもできるが, ゲートウェイ専用の機器としてルーターと呼ばれるものが利用される. ゲートウェイには他の計算機資源と同様, IP アドレスが与えられる. このほかに, 信号をある程度制限しながら LAN の内部の接続をおこなう装置にブリッジと呼ばれる機器がある. ブリッジは IP アドレスをもたない. さらに, LAN 内部の接続をおこなう機器としてリピーターがある. リピーターは信号の内容を全く認識しない. 単なる増幅器である.

1.7.1　計算機資源を Internet に接続する人のために

計算機資源を Internet に接続するためには, 各建物等のネットワーク管理者の許可をもらわなければならない. 地球惑星物理学科では研究スタッフ室が管理をしている.

接続に必要なネットワークパラメターは以下の通り.

国際理学ネットワークの IP アドレスはクラス B で

133.11

である. 理学部はサブネット化して運用しているので, ネットマスクというものを次のように設定しなければならない.

Net / Subnet Mask　255.255.255.0 (または f f f f f f 0 0)

ネットマスクとは, IP アドレスのうちのネットワークアドレスとホストアドレスとの分割を決めるフィルタである. 地球惑星物理学科の各号館ごとに張られている LAN は国際理学ネットワークのサブネットであり, 次のようになっている.

理学部 1 号館	ネットワーク IP アドレス	133.11.25
	ネットワークドメイン名	grl.s.u-tokyo.ac.jp
	ネットワークサーバー名	sungrl1
	ネットワークサーバー機種	Sun ?
理学部 3 号館	ネットワーク IP アドレス	133.11.24
	ネットワークドメイン名	geoph.s.u-tokyo.ac.jp
	ネットワークサーバー名	geoph
	ネットワークサーバー機種	Sun SPARC station 2
理学部 7 号館	ネットワーク IP アドレス	133.11.7
	ネットワークドメイン名	geoph.s.u-tokyo.ac.jp
	ネットワークサーバー名	geoph
	ネットワークサーバー機種	Sun SPARC station 2
理学部 化学東館	ネットワーク IP アドレス	133.11.?
	ネットワークドメイン名	
	ネットワークサーバー名	
	ネットワークサーバー機種	

1.7.2　主要計算機資源アドレス, パラメーター

ネットワーク上の主要計算機資源のリストを上げておく.

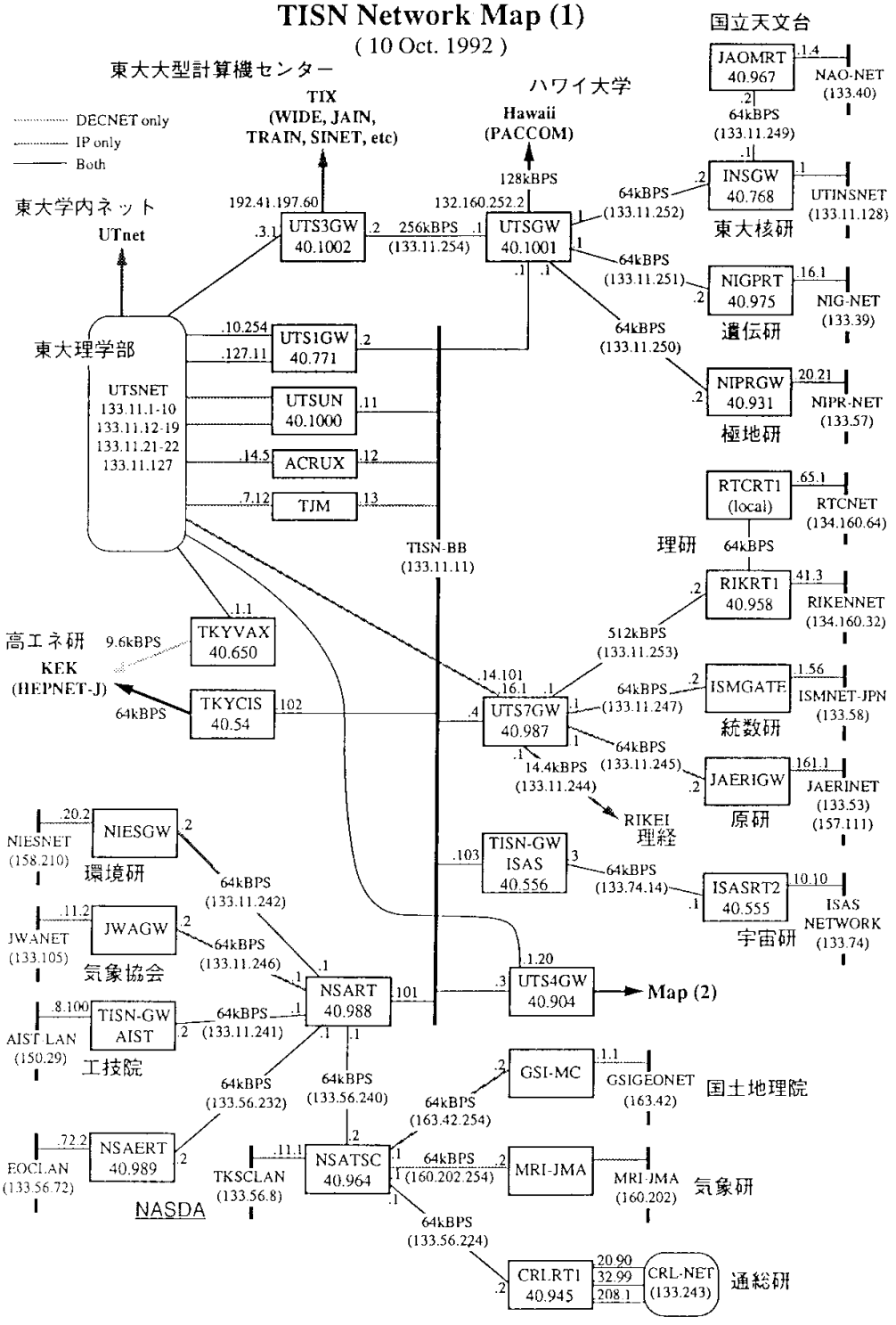


図 1.2: TISN (国際理学ネットワーク) 地図 (TISN ミーティング資料 1992/10 による).

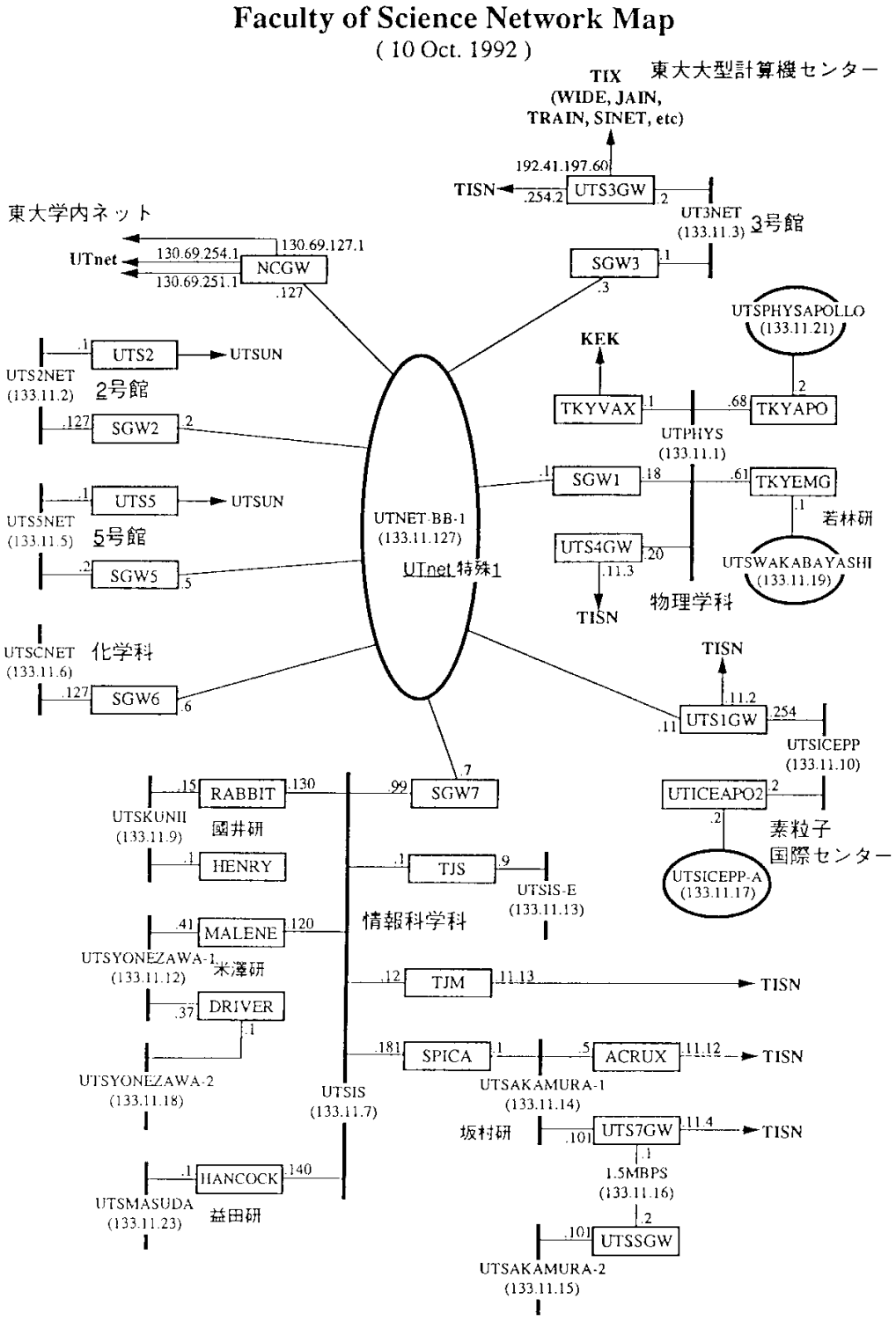


図 1.1: 理学部ネットワーク地図 (TISN ミーティング資料 1992/10 による).

1.7　UNIX / X window / Internet 環境基本用語集

ここでは, 主としてイーサネット (Ethernet) を媒体とした UNIX の世界のネットワーク Internet での基本単語を紹介しておくことにする (上原, 1988 参照)

Internet は米国国防総省高等研究計画局 (DARPA) のプロジェクトに端を発するネットワークシステムの総称名称で, DARPA Internet とも呼ばれ, 計算機間の接続 (ネットワーク) 機構, 並びに, 複数のネットワーク間の接続機構を統合的に与え全体を巨大な一つのネットワークと化するものである. Internet は IP (Internet プロトコル) と TCP (トランスミッションコントロールプロトコル) を代表的なプロトコルとするので TCP/IP Internet とも呼ばれる. プロトコルとは, 日本語で通信規約というが, 信号の符合化のとりきめのことである. DARPA Internet は UNIX BSD4.3 に組み込まれたために UNIX マシン間をつなぐネットワークとして広く世に広まり, 今や世界中の (UNIX 以外のも含めて) 計算機が接続されつつある, という状況をもたらすに至った. Internet 上の計算機は IP アドレスという固有の番地 (133.11.11.11 とかいうやつで米国で管理されている) をもっている. (詳細 は Douglas Comer (村井純 ・楠本博之訳, 共立出版) 『bit 1990 年 7 月号別冊 TCP/IP によるネットワーク構築」). 簡単にいえば, コマンドでいうところの `ftp` や `telnet` が使える世界である.

イーサーネット (ethernet) の語源はその昔光の媒体として考えられた エーテル (ether) である. 現在では, 同軸ケーブルに電磁波を閉じ込めて通信に使ったものに使うことにしたわけである. 最初にイーサーネットをこさえたのは Xerox 社 (1976) ということになっている. 現在, イーサーネットという単語のさすものは, ハードウェアとしての同軸ケーブル (3 号館廊下をはいまわる黄色い線) と, その規格のことである. イーサーネットは 10Mbps の情報転送能力をもつ.

ローカルエリアネットワーク, **LAN** (local area network) とは, 企業内 ・地域情報通信網のことである (何のこっちゃ). この定義では通常の内線電話網なども LAN の一種と言うことになる. しかしながら, LAN と普通に言えば, 計算機資源を結ぶ通信網のことさす. イーサーネットは, LAN を実現する物理的媒体と規格の一つである. もちろん, 内線電話網で LAN を作ることもできる.

プロトコル (protocol) とは, 媒体の上を走る電磁波の規格のことである. 通信規約と訳する. プロトコルとしてよく登場する名称としては, 米国国防総省 (Department Of Defense) の ARPANET(Advanced Research Projects Agency net) 上で開発された, **TCP**(transmission control protocol), **IP**(internet protocol), **ARP**(address resolution protocol), **RARP**(reverse address resolution protocol), などがある.

TCP/IP 通信とは, プロトコルとして TCP と IP を用いた通信方式で, 4.x BSD (Berkeley System Distribution) UNIX の世界で標準的に搭載されたために広く世の中に用いられている. TCP/IP 通信をおこなうためには, 計算機資源は世界中で唯一に決められた住所 **IP** アドレスをもたなければならない. IP アドレスはホストアドレス (ネットワーク上の計算機資源のアドレス) とネットワークアドレスとからなる 4 つの数字の組である. それぞれの数字は 0 から 255 までの値をとる. ネットワークはそのアドレスに用いる数字の数によってクラス A(1 ヶ), クラス B(2 ヶ), クラス C(3 ヶ) と呼ばれる.

TCP/IP 通信を用いた LAN の世界では, ARPA サービスと呼ばれる **TELNET** (仮想端末機能) と **FTP** (ファイル転送プロトコル, file transfer protocol) が, そしてまた, Berkeley サービスと呼ばれる rcp, rlogin が利用できる. これらによって, 遠隔地にある計算機を操作したり, ファイルの転送をおこなったりすることができる.

イーサネットを用いた TCP/IP 通信の世界を構成するハードウェアたちには次のようなものがある. 計算機資源とイーサネットケーブルとは, ケーブル上にトランシーバーと呼ばれる接続口を取り付け, トランシーバーケーブルを用いることによって接続される. トランシーバーケーブルは 2 段階まで分岐させることが可能でそのための装置をマルチポートトランシーバーという. 建物内のイーサネットケーブルの長さは最長で 500m である. それ以上延ばすときにはリピーターと呼ばれる増幅器が必要になる. 建物と建物との LAN の接続には光ファイバーが用いられる. これは建物間の絶縁のためである. 光ファイバーとイー

エディターと日本語フロントエンド (FEP) は何を使ってもよい. とにかくテキストファイルが作れるようになればよいのだ. とは言うものの, DOS マシンならば Vz というエディターと WXII という FEP の組み合せが御推奨である. UNIX のエディター vi や emacs の PC 版も世の中には存在している. マッキントッシュの世界はあまり高速エディットという見地からは軽快な環境ではないようだ.

言語については, 歴史的事情と HITAC (大型計算機センター) のコンパイラの問題で, FORTRAN を公用語にしている. しかし, ワークステーションの公用語は C である. 能力者は, 2 ヶ国語以上喋るのが結構であろう思う. 特に, これから UNIX の世界に突入していくであろうから C は必須である. おいおいスパコンでもまっとうな C コンパイラが動くようになってくるであろうが, 科学技術数値計算の主流はまだまだ (2 ~ 3 年は) FORTRAN であるようだ. 能力のあまり高くないものが無理に 2 ヶ国語に挑戦すると言語障害に陥るので注意. とりあえずは古典芸能, FORTRAN でもって数値計算と簡単な文字処理はできるようになりたい. もちろん, 能力者は適宜使い分けて欲しい.

ドキュメントの制作は研究活動の基本仕事であり, 最終的な財産でもある. したがって, ドキュメントはなるべく計算機の動向には依存しない形, すなわち, アスキー形式のファイルで保存されるようにしたい. そこで強く推奨されるのが TEX による文書処理である. 計算機非能力者であっても TEX が動かせるぐらいにはなってもらいたい.

残念ながら, 画像情報のファイル形式は標準化されているとは言えない状況なので, そのやり方は広く模索中である. よい方法を考えてほしい.

1.6　ネットワーク環境

地球惑星物理学科に引かれている Internet は TISN (東京大学理学部 ・国際理学ネットワーク, Todai International Science Network, 釜江, 1989 参照) の一部を形成している. 地球惑星物理学科では, すべての居室にトランシーバーケーブルが引き込まれ, ハードウェア的にはいつでも利用可能な状態になっている. ただし, 実際に計算機等の接続を行うには, 研究スタッフ室にてユーザー登録をし, IP アドレスを交付してもらわなければならない. 研究スタッフ室には, 地球物理学教室におけるネットワークサーバーとして, UNIX ワークステーション SunSPARCstation 2 (geoph) が設置されている. geoph は研究スタッフ室によって管理されており, 研究スタッフ室に申請することにより login アカウントが交付される. geoph は, 教室構成員に対しては電子メールサービスをおこなう.

理学部ネットワークのネットワークサーバーは, 情報科学科にある SUN-3/280S (utsun) である. 国際理学ネットワークは, 理学部 7 号館に設置されているゲートウェイ装置 (ルーター) P4200 を介してハワイ大学に接続されており, アメリカ, さらに, 世界へ TCP/IP による通信が行えるようになっている.

理学部 3 号館 LAN から他の号館への接続用の器機 (ゲートウェイ装置とモデム, 光トランシーバーなど) はすべて地階工作室に設置されている.

い方ができる. これらとて使いこなそうとすれば結構難しい. パソコンなのであるから最低限自分の環境は自分で維持できるようになってもらいたいものである.

マッキントッシュは米国 Apple 社製のパソコンのブランド名である. キーボードからのコマンド入力を基本とする DOS マシン とは対照的に, マウスを基本としたグラフィックユーザインターフェイス (GUI) に特徴があったのであるが, 最近は MS-Windows の普及によって DOS マシンも似たようなものになりつつあり, 『マックならでは」という先進性は段々薄れてきた. それでも米国起源のソフトウェアの充実度はマックならではの環境を提供してくれる. マッキントッシュはそのとっつきの良さから, 米国ではいわゆる理科系でない世界でも爆発的に広まり, グラフィック ・音楽など色々な分野のプロの使用するところとなってきている. もちろん, 数学 ・数値解析 ・グラフ作成など理科系研究者向けのソフトも充実しており, 大規模大量のデータ処理 ・ソフトウェアのグループ開発を必要としない研究の現場でも大いに活用される傾向にある. しかし, プログラム開発を主体とする研究にはあまり向いていない. Internet にどっぷり浸かった仕事をしなければならない計算屋が使うとすぐに種々の限界にぶつかることになり, このような人々には単なる 『清書機」に過ぎないものになってしまう. マッキントッシュの弱点は UNIX マシンと同様日本語処理機能だと言われていたが, 最近ではこれもかなり改善されている.

DOS マシンはメモリ容量やディスク容量が小さくても動き, 総じて値段も安い. 以前は IBM PC で日本語の世界を扱うことができなかったのだが (コンパチマシンでは日本語 OS と 英語 OS とをいちいちリブートして切替えるというダサイことをやっていた), DOS V になってその必要がなくなった. もはや NEC PC98 も大名商売をしてはいられないであろう. ますます価格が下がることが期待される. DOS マシンでは OS を維持する上で必要なファイル数が非常に少ないので, 最低限のシステム管理のてまや覚えるべきことは少ない. DOS マシンで複雑なシステムを組む場合には標準がない. たとえば, ネットワーク用ソフトは供給元ごとに違うので, ローカルに管理のノウハウを蓄積する必要がある. ハードウェアを直接いじるような応用に関するノウハウは他のコンピュータよりも進んでいる. たとえばデータロガー的使い方. グラフィックスも, 互換性を無視して性能だけを追求するならば, DOS マシンのハードウェアを直接操作するほうがワークステーションより楽である. もっとも難題なのは, 8086 系 CPU の構成からくる MS-DOS のメモリの限界である. 少し複雑なことをしようとすると, 8086 系 CPU 特有のセグメント構成や絶対メモリアドレスを意識しなければならない. いろいろなオプション装置が多くのメーカーから供給されているが, DMA チャネルやインタラプト番号のような資源が有限なために組み合わせ不可能なこともある.

1.5 ソフトウェア

計算機を用いた仕事のなかでも特に数値データの生成とその処理に 重きをおいた場合, 自分の計算機でこなすべき仕事は次の 5 つにまとめられる:

- プログラムの編集能力をもつ
  (高速エディターがインストールできて動かせる)
- 多少の計算能力をもつ
  (言語コンパイラがインストールできて動かせ, プログラムの虫取りができる)
- 文字処理能力と作画能力をもつ
  (ドキュメントの作成, さまざまな情報データの処理ができる)
- スパコン等他の計算情報資源とのファイル転送ができる.

実際, Internet に接続されたパソコンを用いてもこれらの仕事をそこそこなすことができる. 作画能力はマッキントッシュや DOS マシンが有能である. しかし, 計算仕事をするならば UNIX でこれらの作業ができるようになってもらいたい.

ばならない.

1.3　計算仕事をやっていく人のために

現在, 地球惑星物理学教室の住人の大半は何らかの形で計算機を武器として用いることにより仕事を行っている. しかしながら, 小数の例外を除けば, そのほとんどの人間 (特にスタッフ) の計算環境整備能力レベルはあまり高いとは言えない. 特に, 計算機と OS に関する知識はほとんどまったくない. これは, ひとえに大型計算機センターという神様の存在に おんぶにだっこしていたことによっている. しかし, 先に述べたように, 世界が分散環境の時代に入り大型計算機センターの陣痛力が薄れてきた今日, 技術進歩に歩調をあわせていくことは自分でやらなくてはいけない.

さて, 非計算機業界において, 計算機を使った仕事をするプロとして商売をしていくためには次のような 2 項目をクリアーすることが必要となる:

- 計算機の基礎知識をもつ:
  分散処理時代で, 自分のマシンをもってやっていかねばならない. そのためには自分で計算環境を構築できるに十分な計算機と OS とに関する基礎知識をもたなければならない. さらに, 分散処理時代における, 世界のソフトウェア生産レベルに追い付いていかなければならない. スパコンと自分の計算環境をうまく結合させる必要がある.

- アプリケーションソフト製造能力をもつ:
  計算機業界が高度化多様化し, プロの計算屋はアプリケーション人の面倒などいちいち見てくれない. プロのソフト屋はコンパイラと OS しか作らない. 売れ筋ソフト, または, 非常に大口の注文しか受けない. 仮によいソフトがあってもインストールの手間まではかけてくれない. ユーザーインストールが基本である. 専門家がケアしてくれなくなったこのような部分を自分で埋めなくてはならない. 最適のアプリケーションソフトを自作あるいはさがしてきて自分のマシン上にインストールできないといけない.

90 年代にはいってからの世の中の変化はさらに目覚しく, 分散環境はより進み, スパコンまでもがカスタマイズされる御時世となりつつある. 特定の計算に応じて特定のチップを設計する, などということが一昔前に比べれば圧倒的に安価にできるようになってしまったので, 計算をするのにソフトウェアでなくてハードウェアをいじらなくてはならなくなってしまう人々も出現している. 大昔に戻ってしまったわけである. これらの動向にどう対処して行くか, ここから先は実験型計算屋と理論型計算屋とで指向がわかれるところである. どれを選ぶかは得意不得意で選択すべきであろう.

1.4　計算仕事はあまりやっていかない人のために

UNIX とその周辺, 特に Intenet をめぐる世界は避けては通れないのでこれだけは言葉ぐらいは常識として知らなければならない. それ以外は, カホウは寝て待て, で計算を仕事にするやつらが勝手に環境を整えてくれるのを待っているというのも手である. そのような人でも, しかしながら, 計算機による情報処理をやらないわけには行かない. 計算機のことなどあくまで考えたくないというのならば UNIX / X window / Internet についての最低限の知識を習得した後は, DOS マシンあるいはマッキントッシュの世界にこもるのがよろしかろう. これらのマシンは研究系ばかりではなく, 事務系や趣味の世界でも (の方が) 普及している. したがって世間一般と同じ種類の仕事に使うならば流通しているソフトウェアが多く, かつ, 安価である. 構造が単純で, リセットをかけたり電源を落としても被害が少ない. そのためかなりあらっぽい使

高速計算機が自動的に 設置されるわけにはいかなくなってしまった. おかしな話だが, 我々は常に日米間において分が悪い方の計算機を買わされるという, 政治的に低い地位にある. 日本の計算機が劣悪なときには日本製を, アメリカの計算機が劣悪なときには アメリカ製を使わなければならない. 我々が欲しい米国製の計算機はワークステーションなのであってスパコンではない, がそれはもちろん構造協議の対象にはならない.

かくして, 大型計算機センターは計算機利用者のための支配者にして偉大なる守護神としては君臨できなくなっているのである. 実際, 計算機センターなるものの今日的な地位をもっとも端的に表しているのは, 計算機業界で有名な MIT の状況であろう. MIT と言えば, Athena project (キャンパスネットワーク) や X-window, AI Lab. に代表されるような, 計算機科学の先端的な大学の一つであるが, この大学のもっている共同利用のメインフレーム計算機は, はっきり言ってどうしようもなくとろい (ただし, これは 87 年当時の話. ちなみに, そのころ MIT は NEC のスパコンを入札したが, 連邦政府より待ったがかかって, 結局スパコン自体を買えなかった. 計算機業界人ではない普通のエンドユーザーにはあまり住みよいところではない. 今ではもう少しましになっているだろうけど). MIT のような大学, あるいは日本の類似の研究機関でも, 開発は汎用スパコンの先をいって, 専用回路により計算の高速化を図るような方向にどんどん進んでしまっている. すなわち, 特定の種類の計算専用のハードウエアを開発することにより, 小型ながらスパコン並あるいはそれ以上の性能を実現する, という方向である. エンドユーザーにとってはわけがわからん世界なのであるが, 留まるところを知らない.

専用回路の設計にまで踏み込むことはとりあえずおいておくとして, 今日, 数値計算を仕事として行なうためには, スパコンによる実行能力の高さと, ワークステーションによる開発能力, ソフトウェア保守能力の高さとを同時に追及しなければならない. 高い能力が利用可能であるということは, 逆に, それを利用し生産の合理化を計らねば生き残ってはいけないと言うことをも意味する. 計算機センターをめぐる政治的状況はあまりよろしくはない. 計算機ユーザー全体に対する計算機センターの地位の低下は著しい. 数値計算を仕事としない人々にはもはや計算機センターは無用の長物である. しかし, 計算仕事をする人のスパコンへのニーズはますます大きくなっており, 計算機センターは, これからもスパコンセンターとして我々には不可欠な存在としてその地位をたもつであろう. そのような人々は, スパコンセンターとしての計算機センターの存在を側面から支援しなければならない. 米国では長い間汎用計算機 / スパコンセンターに集中投資が行なえない構造になっていたのであるが (もちろん国防予算の集中投資を受けて Cray や Internet はできたわけだけれど), 日本のスパコン計算機進出におそれをなしたレーガン政権の後期に全米数箇所の日本型スパコンセンタが設立され, ソフトウェア資源の集積を始めとして日本のセンタ (基本的にハードウェアがあるだけ) を大きく引き離す活動が行なわれるようになってきている. 一方, 我々は自分のまわりにスパコン以外の計算機資源を配置し, インストールし, スパコンと接続し, 保守し, 器機の更新をするという一昔前ならば計算機センターが全部やってくれた ことを自分自身で行わなければならない. したがって, もはや我々ユーザーにも 秒進分歩の計算機業界の動きにある程度ついていかなければならないという厄介な作業が発生してしまったのである.

地球 (惑星) 物理学科では, ようやく 1989 年にはいってイーサネットが敷設され Internet の世界に突入し, 大型計算機センターなどと TCP/IP 通信ができるようになった. 現在地球惑星物理学科の研究室にあるほとんどすべてのパソコン, ワークステーションは TCP/IP でのネットワーク通信を行えるように整備されている. 気象グループでは早くから大学院生を含め一人一台 PC 化を進め, 1990 年始めにはほぼ全員がパソコン (残念ながら UNIX ワークステーションではなかったが) による高速編集と TCP/IP でのネットワーク通信を享受できるようになった. この環境は 1992 年に立ち上がった気候システム研究センターに受け継がれ, そこでは一人一台ワークステーションの世界が実現されるに至っている. 地球惑星物理学科では, ようやく環境の UNIX 化が始まったところであり (3 号館では何と 1992 年) 一人一台のワークステーション環境までにはまだ程遠い. それでも順次整備が続けられていくであろう. 生産活動においては, これらを有効に使いこなせることが必要となる. 同時に, これらを保守, 管理, 更新していくことができなけれ

ターではなくて別な世界を開くのに忙しかっただけのことである. 日本が国策でもってスパコンを作っておったころ, 米国ではいわゆる UNIX ワークステーションとネットワークを組み合わせた世界が実用的に完成し, 価格が下がり, 研究者の間に急速に広がりつつあったのである. 計算機をネットワークでつなぐ試み自体は, たとえば, DEC 社の製品 VAX/VMS の世界では, DECNET, IBM のマシン間では BITNET, 同様にして日本の大学計算機センターの汎用大型機間では N1NET, という具合に広まり, 同じ技術を使った商用電子掲示版 OMNET とかが開設されるようになってきていた. 古くから VAX を使っていた高エネルギー物理, 天文, 宇宙空間物理のグループは DECNET で会話し, 海洋グループは OMNET を利用したりしている. しかし, UNIX に実装された Internet の世界は, そして, X-window の世界は, 接続された他の計算機をあたかも自分の計算機と同一視して使えるという大きな特色を持ち, 計算資源の有機的結合を可能にしたのである. 研究者各人は一人一台ワークステーションをもち Internet でもって結合する, それでいて全体は一つの巨大システムのごとく振舞う, という分散環境時代の到来である. 分散処理にし, 一人一台の CPU を持つことにより, メインフレームの重たい OS にのっかったファイル編集では考えられない軽快なソフトウェア開発を可能にしてくれたわけである. しかも, ワークステーションは情報科学屋の世界 (UNIX がよく使われる世界) で開発された機械であるだけに, ソフトウェアの生産効率を最適化するように作られている. さらにまた, ビットマップディスプレイとウィンドウシステムの開発はソフトウェアの生産効率を著しく向上させた. 80 年代も終りになると一人一台ワークステーションのパワーの大きさがシロートさんにも明かになる. スパコンによる実行能力の高さと, ワークステーションによる開発能力とソフトウェア保守能力の高さとを Internet によって結合し, 同時に追及するようになってきたわけである.

80 年代に起こったもう一つのながれは, いわゆるパソコン (DOS マシンやマッキントッシュ) の台頭である. 計算機センターにしか計算機がなかった時代は終りを告げ, 誰も彼もがそこそこそこリッチであれば計算機を個人で持てるわけである. パソコンの台頭がもたらしてくれたものは, 計算機の計算からの脱却である. パソコンは多くの人にとっていわゆる計算のための道具ではなく, ワープロであり住所録であり, 予定表なのである. 計算機は長い間非常に高価であったので, いわゆる情報科学をやっている人以外このような使い方は許されなかったのであるが, それが一般利用者のもとにおりてきたわけである. 計算機は単なる文房具になってしまったのである. もちろん, 日本において計算機が文房具になるためには, 計算機が日本語を喋れるようにならなければならなかったのであり, それは 1980 年代にようやく実用化されたのである. 英語世界においては計算機ははやくからワープロとして機能しており, UNIX や VAX の VMS ではそのための道具が基本ソフトとしてのっかっている. UNIX や VAX/VMS が日本でしばらくはやらなかったのは, 金がない, あるいは, 通産省等の国産化指導もさることながら, 欧米諸国でうけていたこのワープロ機能が役にたたなかったのが大きな原因の一つである.

Internet を使った分散処理, 一人一台の計算機の普及は, 大型計算機センターの存在そのものにも影響を投げかけつつある. 70 年代から 80 年代前半ならば計算機センターに集まらざるをえなかった計算機ハッカーたちは, 今や皆自分のマシンをもち, ネットワーク上に分散して集うようになってきた. 以前は, 大型計算機センターの OS, VOS3, にのるような道具を作り, 大型計算機の使い方を教えるような立場になっていたこの手の人材はそこにはもはや集まらなくなってきているのである. 計算業界の主戦場も, 大型メインフレームの開発がひとくぎりついた今日では, 分散環境への対応に移行しており, 汎用大型計算機のハードウェアにもソフトウェアにも昔のようには技術と人の投入が行われなくなってきている. 計算機業界は急速に多様化高度化してしまい, 大型数値計算というあまり面白みがない (新味がない, またはもうからない) 世界, 特にそのアプリケーションソフト開発には, 人がはりつかなくなってしまった. 大型計算機上で動くアプリケーションソフトは総じてダサイものが多く, DOS マシン や Macintosh の軽快なソフトには遠く及ばない. 一方, ワークステーションの世界では, RISC (reduced instruction set of computer) チップの開発により, 数十 MIPS (million instruction per second) は当り前という御時世になってしまって, 汎用計算機の威力はますますくすんできている.

さらに, 追い打ちをかける事態として日米貿易問題があって, もはや, 東京大学だからといって日本の最

計算機センターによるこのような計算機の使い方は, 計算機を実験・観測の制御・処理に使う人たちの使い方とは大きくことなる. 後者では計算機は小人数のグループで占有して使うため, 機器の整備・更新はグループの構成員が行なうべき当然の仕事であり, 従って, 少なくとも研究グループの身内の誰かが機器に精通していなければならない. もちろん, 単なる計算だけのために計算機を小数のグループで占有して使える金持ちなグループでは, 事情は実験・観測のための利用と同じことになるのではある. しかし, 特に汎用計算機は非常に高価であったため, 通常の科学技術計算を行ないたいだけの一般利用者がそのような立場に至ることはあまりなかったと考えて良い.

日本では, 大学での計算機の利用は, 拠点大学におかれた計算機センターの汎用計算機を全国共同で用いる形態が基本であった. 実際, 計算機学科以外では高エネルギー物理学など特定の重点投資分野でなければ一般利用者は貧乏で大型計算機などを小数のグループ (例えば研究室や学科) で占有して持つことなどほとんど不可能であったといってよい. このことは, 逆説的ではあるが, 利用者には 3 つの点で大きな利益をもたらした. まず第一に, 利用者はハードウェアとしての計算機資源をもたない (もてない) ので, その面倒を見ること (インストールや保守, 機器更新) もしなくてよい. それらの仕事は計算機センターで行ってくれたわけである. 一般利用者は, 計算機の使い方の修得とプログラミングを行うこととに専念していればよい. 第二に, 共同利用の計算機センターは小数の拠点大学にしかなく, したがって, 投資は自動的に集中され, 国産装置としては最新の計算環境が, そこにでかけさえすれば, ほぼ自動的に手にはいるようになっていた. 第三に, 計算資源の集中の結果, 人と情報も強制的に計算機センターに集められることになり, 情報の交換が容易であった. 大型計算機センターの設備は常に最新のものに更新され, かつ, 他のどの場所にもないので, そこに人々が集中し, あらゆるレベルの人間がその場で仕事をすることになった. したがって, 初心者にとっても熟練者にとっても情報交換は非常にたやすい. いいかえると, 計算機の利用方法をあまり知らなくても, 現場にいけばなんとかなるわけである. 大型計算機センターが偉大なる守護神として君臨していたわけである. 計算環境の詳細は大型計算機センターが考えてくれる, あるいはそこに情報が集約されているから, 人々は, 計算機などには関係しない業界固有の問題を追及すればよかったのだ.

計算機センターにでっかいホストがあって, 教室には実験観測用マシンを除いては端末程度の資源しかないという状態は 80 年代を通して不変であった. 米国では当然ずっとリッチな環境であり, 学科単位ぐらいでちょっとした汎用計算機 (特に IBM コンパチのメインフレームと呼ばれるものではなく DEC 社の VAX という名の, より便利な OS – VMS – を備えた計算機) を持つことが普通になっていった. しかし, 面白いことに, なまじっか金持ちであったことが災いして, 数値計算の分野では逆転現象を生むことになってしまったのである. 日本では, 幸か不幸か, 80 年代前半の計算機業界のスパコン技術開発競争と拠点大学には自動的に国産の最高速計算機が導入されるという政府通産省・文部省のスパコン開発計画のお蔭で我々はほとんど何の努力をすることもなく, 自動的に世界最高レベルのスパコンが使えるという結構な身分になったわけである. 米国では金持ちなので VAX は買えたけれどスパコンを購入しようとすれば非常に大変な努力を要する. 日本では貧乏なので端末ぐらいしか買えないけれどもスパコンはある. ちなみに, それ以前は, 他の昔の日本工業製品と同じく, 国産計算機というダサイものを無理やり使わされて不利益をこうむっていたわけである. 幸にして, これまた他の工業製品と同じく日本のスパコンは今や米国から目のかたきにされるほどよくできている

というわけで, ラッキーにも我々は 80 年代後半には, 地球・惑星科学業界の大学関係研究教育機関としては「世界でもっともでっかくて速いスーパーコンピューター」を身近にもっている場所の一つになった. 実際, 例えば, 本学科の気象グループ, 海洋グループともに 87 年ごろには, 計算機使用料や現場のソフト開発能力を除けば, 米国の国立研究所である GFDL(地球流体力学研究所, プリンストン大学/米国大気海洋庁) や NCAR (米国国立大気研究センター) という業界老舗の人たちに自慢できるほどの計算機ハードウェア環境に接していた. しかも, その計算機資源のインストール, 保守や器機更新などの雑務は計算機センターが完全にこなしてくれるので, ユーザーに徹しておればよいという真に結構な状況であったわけだ.

さて, 米国の計算機業界人も遊んでいたのかというとそういうわけではない. 汎用スーパーコンピュー

後, 仮に UNIX / X window 環境に直接お世話になることはなくとも, 電子メール / ネットニュースと切り離されて仕事を続けることは, 少なくとも研究業界にいる限り, あり得ない. 電子メール / ネットニュース, そして, Internet は市中においても急速に広がりつつあるので科学・技術的な仕事にたずさわる限りいずれその波に取り込まれることになるであろう. 個々の仕事をマッキントッシュや DOS マシン, あるいはその他の専用マシンでこなしている人でもこれら Internet のパラダイムからは無縁ではあり得ないのである. 実際, 多くの研究室で, パソコンは Internet に接続されており, 電子メールはいうに及ばず, 諸々の情報通信がネットワークを介して行なわれている.

現在のカリキュラムでは, これらの環境の構築整備のための授業等々は行なわれない (あるいは, 行なえない). しかしながら何らかの計算機・情報処理装置を主要な道具として仕事を行なうつもりの人はいずれ環境整備・構築を自らが行なわなければならなくなることを覚悟してもらいたい. 自分の使う道具は自分で整備できる, という当然の前提を別にしても, UNIX / X window / Internet 環境が「急速に広まりつつある」その主体が諸君に他ならないからである. これらの環境はほっといても勝手にやってくるわけではない. 先ほど「常識」と書いたけれどもこのような環境が常識になったのは日本の特に地球惑星物理の世界では 80 年代も終りにきてからなのであり, まだまだ,「文明開化」していないところもたくさんあるからである. 実際, 現在, 日本の Internet のネットワークはそのほとんどが若手有志によるボランティアネットに過ぎない. UNIX / X window / Internet 環境なんてものはいまどき当たり前に存在してないといかんのや, と思った連中が勝手につなげてきたものが実態であったりするのである. 事情は時間とともに改善されていくであろうけれども, まだ当面は諸君ら自らが手に汗して環境を作る時代が続いていくことであろう.

というわけで, いつなんどき環境構築をしなければならないかも知れない諸君には, 少なくともいまどきの計算環境の概略に触れていただき, いざという時にびっくりしないでいられるようになっておいていただこう, というわけである. また, そのような裏方仕事の存在に多少は気を配っていただき, 先輩諸氏の努力に対して「お百姓さんありがとう」と自然にとなえられるようになっていただこうというわけである. 少なくとも, さまざまな人々によって作られてきた公共的ソフトウェア資源, あるいは, Internet 等の公共的ハードウェア資源など, 計算環境のを構成する基本的資源を利用する際のお作法はしっかりと身につけていただき, 今日の諸々の計算環境を維持してくれている見えない人々に迷惑をかけないようになっていただかなくてはならない. 強調しておかなければならないことは, 多くのシステムはボランティアで運営されているということであり, 従って, その利用に際しては十分な注意と敬意を払わなければならないということである.

1.2　計算・情報処理環境の歴史と一般利用者の責務

電子計算機の誕生のころは, 計算をしようと思う者は計算機について知りつくし, かつ, よく面倒をみてあげなければならないことは当然であった. しかし, IBM の計算機に代表される, 汎用 (大型) 計算機 (メインフレーム = main frame) が登場するにいたり, 計算をしようとするものは計算機のことを (あまり) 知らなくてもよくなり, かつ, 面倒を見る必要もなくなっていった. 汎用大型計算機の特色は複数の利用者によって利用されるということであり, したがって, 誰かが管理してくれさえすれば, 残りの利用者はややこしい裏方仕事は全く知らなくて良く, 単なる使い方に習熟していれば良いことになったわけである. 実際, 汎用大型計算機は装置の専従管理人が存在することを前提とし, 計算機を導入するということはそのための人をやとうということとほぼ同義であったので, この用件は自動的に満たされている場合がほとんどであった. このような計算機の利用方法を計算機センターによる利用といい, 通常の科学技術計算を行ないたいだけの利用者は当該機関の計算機センターにある計算機を利用する, というのが長いあいだのならわしであった. もちろん誰かが努力して計算機センターを作らねばならないけれど, 誰かが作ってくれていればしかるべき利用料などを払うことによりサービスが受けられたわけである.

第 1 章

東大地球惑星物理学科周辺の計算機環境の変遷

1.1 計算・情報処理環境に対する態度

ここでは, 東京大学理学部地球惑星物理学科近傍における計算環境整備変遷の歴史, ならびに, 計算・情報処理技術教育の基本的ポリシーについて述べる. 基本的ポリシーは一言で言えば次のようになる:

> 地球惑星物理学において必要となる計算環境の利用技術, さらに, 計算環境の構築整備技術を身につける.

そのための武器と用い方は

> 大型計算機センターのスパコンによるスーパー計算, 特定の実験, 観測, 計算のために設計された専用機による処理, そして, 個人用マシンによる情報処理

である. 計算環境は利用できるだけでなく構築もできなければならないことに注意しよう. 個人個人が情報処理のプラットフォームを持たなければならない時代になってしまったのである. 自分の使う道具ぐらいは自分で整備できるようにならなければならない. もちろん, 道具はそれぞれの人がそれぞれの必要に合わせて選べば良く, かつ, 利用・整備技術もそれぞれのレベルで身につければ良い. 以下で述べるようにカリキュラムとしては UNIX ワークステーションをそのような個人用情報処理プラットフォームとして想定している. しかし, 計算のプロになる人以外の多くの人は実際にはパソコンをプラットフォームとして選ぶことになるであろう. 価格の問題もあり UNIX ワークステーションを個人で購入するのはちょっと難しい (車を買うことを思えばコンパラだが).

本演習では UNIX / X window / Internet 環境 (後述) に習熟し最低限の利用方法を身につけてもらうことを目的の一つとしている. 地球惑星物理学関連の個別的な仕事が UNIX / X window / Internet 環境を必須としているわけでは必ずしもない. むしろ, パソコン (DOS マシンやマッキントッシュ) の進化によって, 数年前に比べれば驚くほど多種多様な仕事が低価格, 高機能にこなせるようになってきている. しかしながら, それでもなおすべての学生諸君にカリキュラムとして UNIX / X window / Internet 環境への習熟を要請しているのは, UNIX / X window / Internet 環境が今や科学技術計算・情報処理の分野ではほとんど 「共通言語」と呼べるようなものになってきているからである. UNIX / X window / Internet 環境以外の環境において使われている諸々の道具類も UNIX / X window / Internet 環境でのそれがモデルとなっているものが非常に多く, あるいは, それに統合されつつある. したがって, UNIX / X window / Internet 環境においての言葉を習得しておけば必要最小限の言語知識が得られることになるわけである. また, 今

6.2.2 tgif+ . . . 125
6.3 xgraph . . . 125
6.3.1 折れ線グラフ . . . 125
6.3.2 印刷 . . . 127
6.3.3 拡大 . . . 128
6.4 gnuplot . . . 128
6.4.1 関数のグラフを描く . . . 128
6.4.2 離散データをプロットする . . . 129
6.4.3 タイトルを付けてみる . . . 129
6.4.4 プリンタに出力する . . . 130
6.4.5 グラフを LaTeX の文章中に取り込む . . . 131
6.4.6 さらに詳しい説明について . . . 133

7 謝辞 **134**

8 著作権 **136**

5.2.5 文字型変数とその他の変数の違い . . . 84
5.2.6 フォーマットの動的変更 . . . 85
5.2.7 多次元配列の記憶順序 . . . 86
5.2.8 サブルーチンへのデータ引渡 . . . 86
5.2.9 EXTERNAL 文と INTRINSIC 文 . . . 87
5.2.10 COMMON 文 . . . 88
5.2.11 その他 . . . 88
5.3 コードの内部表現と FORTRAN . . . 89
5.3.1 整数の内部表現 . . . 89
5.3.2 実数の内部表現 . . . 90
5.3.3 文字の内部表現 . . . 92
5.3.4 ファイルの構造 . . . 96
5.4 最低限の C 入門 . . . 99
5.4.1 プログラムを書く . . . 99
5.4.2 コンパイルし実行する . . . 100
5.4.3 最も簡単な入出力 . . . 100
5.4.4 C 言語の古い規格と新しい規格 . . . 103
5.5 少し進んだ C 言語の話題 . . . 103
5.5.1 C コンパイラの呼び出し方 . . . 103
5.5.2 他処理系から移ってくる方のために . . . 104
5.5.3 ライブラリ . . . 105
5.5.4 lint . . . 106
5.5.5 make . . . 106
5.5.6 emacs の C mode . . . 108
5.6 FORTRAN と C の比較 . . . 109
5.6.1 FORTRAN と C の言語仕様の比較 . . . 109
5.6.2 FORTRAN と C の言語の環境の比較 . . . 109
5.7 良いプログラムの書き方 . . . 110
5.8 PostScript . . . 115
5.8.1 PostScript の書き方 . . . 115
5.8.2 特徴 . . . 115
5.8.3 サンプルプログラム . . . 116

6 文書の作成と描画 120
6.1 TEX& LATEX . . . 120
6.1.1 ソースを書く . . . 121
6.1.2 LATEX 処理をしてみる. . . . 122
6.1.3 xdvi を使ってのプレビュー . . . 122
6.1.4 プリンタへの出力 . . . 122
6.1.5 MS–DOS で作ったソースをコンパイルするときは . . . 123
6.1.6 うまくいかないときのヒントをいくつか . . . 123
6.1.7 LATEX 文章に絵を取り込む方法 . . . 124
6.2 draw tools . . . 125
6.2.1 idraw . . . 125

3 ウィンドウ ・システム **45**
3.1 X-Window . . . 45
3.1.1 X-window の起動 . . . 45
3.1.2 X-Window の終了 . . . 47
3.1.3 twm . . . 47
3.1.4 初期画面の説明 . . . 47
3.1.5 root Window でマウスのボタンを押してみると... . . . 49
3.1.6 ウィンドウの操作 . . . 53
3.1.7 カスタマイズの仕方 . . . 54

4 計算機ネットワーク **56**
4.1 Domain Name System . . . 56
4.2 電子メールの使い方 . . . 58
4.2.1 電子メールの仕組み . . . 58
4.2.2 電子メールアドレス . . . 58
4.2.3 電子メールを使う時の注意 . . . 58
4.2.4 参考文献 . . . 59
4.2.5 mh-e の使い方 . . . 59
4.2.6 メールの転送について . . . 60
4.2.7 メールアドレスの別名について . . . 61
4.3 Network News . . . 61
4.3.1 Network News とは . . . 61
4.3.2 学外のニュース . . . 61
4.3.3 教育用計算機センターのニュース . . . 62
4.3.4 ニュースを読むには . . . 62
4.3.5 アクセスできるニュースサーバー . . . 64
4.3.6 簡単なマニュアル . . . 64
4.4 Network 関連命令 . . . 68
4.4.1 ftp . . . 68
4.4.2 anonymous ftp . . . 69
4.4.3 finger . . . 69
4.4.4 talk & write . . . 71

5 プログラム言語 **73**
5.1 最低限の FORTRAN 入門 . . . 73
5.1.1 プログラムを書く . . . 73
5.1.2 コンパイルし実行する . . . 74
5.1.3 最も簡単な入出力 . . . 74
5.1.4 コンパイルとは . . . 78
5.2 FORTRAN77 規格について . . . 78
5.2.1 FORTRAN の方言 . . . 79
5.2.2 不定の概念 . . . 80
5.2.3 ENTRY 文 . . . 82
5.2.4 地球流体電脳ライブラリ GLPSET/GLPGET の構造 . . . 83

2.4 FAQ (Frequently Asked Questions) . . . 26
2.4.1 core . . . 27
2.4.2 #FOOBAR# . . . 27
2.4.3 foobar~ . . . 27
2.4.4 ファイルが消せない. . . . 27
2.4.5 プリンターのキューが消えない . . . 27
2.4.6 良く出てくる “foo” って何ですか? . . . 28
2.4.7 path に入っているのに実行できない . . . 28
2.5 教育用計算機センターでのプリンターの使い方 . . . 28
2.6 shell . . . 29
2.6.1 Shell 変数 path . . . 30
2.6.2 ファイルのモード . . . 30
2.6.3 ハッシュ . . . 31
2.6.4 変数 . . . 31
2.6.5 background job . . . 32
2.6.6 .cshrc . . . 32
2.6.7 .login . . . 32
2.6.8 tcsh . . . 32
2.7 man と Info . . . 33
2.7.1 man . . . 33
2.7.2 Info . . . 35
2.8 pager . . . 35
2.8.1 less . . . 35
2.8.2 more . . . 37
2.9 emacs . . . 37
2.9.1 キー操作表記の約束 . . . 37
2.9.2 概念 . . . 38
2.9.3 emacs の起動 . . . 38
2.9.4 emacs の終了 . . . 38
2.9.5 ファイルのセーヴ . . . 39
2.9.6 ファイルの読み込み . . . 39
2.9.7 今いるバッファにファイルを挿入する . . . 39
2.9.8 命令の中止 . . . 40
2.9.9 カーソルの移動方法 . . . 40
2.9.10 文字列の検索 . . . 40
2.9.11 編集 . . . 40
2.9.12 再表示 . . . 41
2.9.13 バッファの切り替え . . . 41
2.9.14 複数のウインドウ . . . 41
2.9.15 日本語変換の方法 . . . 42
2.9.16 その他 . . . 44

目 次

1 東大地球惑星物理学科周辺の計算機環境の変遷 **1**
1.1 計算・情報処理環境に対する態度 . . . 1
1.2 計算・情報処理環境の歴史と一般利用者の責務 . . . 2
1.3 計算仕事をやっていく人のために . . . 6
1.4 計算仕事はあまりやっていかない人のために . . . 6
1.5 ソフトウェア . . . 7
1.6 ネットワーク環境 . . . 8
1.7 UNIX / X window / Internet 環境基本用語集 . . . 9
1.7.1 計算機資源を Internet に接続する人のために . . . 12
1.7.2 主要計算機資源アドレス, パラメーター . . . 12

2 UNIX ワークステーションの使い方 **14**
2.1 地物標準環境 . . . 14
2.2 とりあえず使えるために . . . 14
2.2.1 表記・その他の約束ごと . . . 15
2.2.2 トラブルに陥ったとき . . . 15
2.2.3 login . . . 15
2.2.4 logout . . . 16
2.2.5 パスワードの重要性 . . . 16
2.2.6 地物標準環境のインストール . . . 17
2.2.7 X-Window の起動 . . . 17
2.2.8 X-Window の終了 . . . 17
2.2.9 kterm . . . 18
2.2.10 shell からのコマンド . . . 19
2.2.11 エディタ emacs . . . 21
2.2.12 エディタ vi . . . 23
2.2.13 C で簡単なプログラムを作る . . . 23
2.3 トラブルに落ちた時 . . . 25
2.3.1 キーボードからの中止の命令 . . . 25
2.3.2 画面出力の再開 . . . 25
2.3.3 よくある, 終了の儀式 . . . 25
2.3.4 殺し屋を雇う . . . 25
2.3.5 Destroy Window . . . 26
2.3.6 他の計算機から入る . . . 26

地球惑星物理学演習

— 計算情報環境入門 —

東京大学理学部地球惑星物理学科
地球惑星物理学演習テキスト編集グループ

1995 年 1 月 25 日 (Ver.1.3)