ISSN 0915-6348

NATIONAL INSTITUTE FOR FUSION SCIENCE

シミュレーション科学教育講座 2003 テキスト

Text of Simulation Science Open Lecture 2003

(Received - Mar. 23, 2004)

NIFS-PROC-55 Apr. 2004

RESEARCH REPORT
NIFS-PROC Series

TOKI, JAPAN

シミュレーション科学教育講座 2003
テキスト

Text of Simulation Science Open Lecture 2003

Key word: simulation science, MHD simulation, PIC simulation, Virtual reality, Visualization

Abstract

Simulation Science Open Lecture 2003 was held October 14-16, 2003 at Theory and Computer Simulation Research Center, National Institute for Fusion Science(NIFS). Unix and Fortran were introduced at the Lecture. The MHD simulation, plasma particle simulation, and visualization for numerical data were lectured, and CompleXcope (virtual reality system) tour was included. Twenty participants attended.

はじめに

この冊子は平成１５年１０月１４日から１６日まで核融合科学研究所理論・シミュレーション研究センターにおいて行われた「シミュレーション科学公開講座 2003」のテキストです。

これまで核融合科学研究所理論・シミュレーション研究センターでは、シミュレーションサイエンスの普及をめざして、４回にわたり「シミュレーション科学教育講座」を行って参りました。第５回となる今回は総合研究大学院大学教育促進事業の支援をうけ、参加者が一人一台のパソコンを利用した実習ができる環境を用意することができました。遠方からを含む約２０名が参加し、Unix と Fortran の基礎(大谷寛明、高山有道)、磁気流体シミュレーション入門（堀内利得)、プラズマ粒子シミュレーション入門（石黒静児)、データの可視化(AVS)（KGT）の講義と実習、CompleXcope(没入型バーチャルリアリティ装置)（田村祐一）の体験を行いました。

このテキストが、これからシミュレーションを始める皆様にとって助けとなれば幸いです。

平成１６年３月

核融合科学研究所
理論・シミュレーション研究センター
岡本正雄

目 次

「シミュレーション科学教育講座2003」 プログラム

日時： 平成１５年１０月１４日から１６日
場所： 核融合科学研究所 研究II期棟４階会議室

１０月１４日（火）
１３：４０～１４：００ 趣旨説明
１４：００～１７：３０ 「UNIX と Fortran の基礎」 太谷寛明・高山有道

１０月１５日（水）
９：００～１２：００ 「磁気流体シミュレーション入門」 堀内利得
１３：３０～１７：３０ 「プラズマ粒子シミュレーション入門」 石黒静児
１８：００～２０：００ 懇親会

１０月１６日（木）
９：００～１２：００ 「データの可視化 (AVS)」 KGT
１３：３０～１４：３０ VR ツアー

第１章での記号法について

第１章では，以下のような記号を用いて説明する．

- sample の書体は，UNIX からの表示を表す．
- `sample` の書体は，UNIX に入力する文字列を表す．
- *sample* **サンプル** の書体は，UNIX に入力する変数を表す．
- *[ ]* の書体の括弧の中は，省略可能なことを表す．
- ␣ は，スペースを表す．
- ⏎ は，キャリッジ・リターン等のキーを表す．
- Ctrl は，コントロールキーを表す．
- Space は，スペースキーを表す．
- Esc は，エスケープキーを表す．
- Delete は，デリートキーを表す．
- BS は，バックスペースキーを表す．
- Tab は，タブキーを表す．
- Alt は，オルトキーを表す．
- Home は，ホームタブキーを表す．
- c は，文字 c のキーを表す．
- Ctrl + c は，コントロールキーと文字 c を同時に押すことを表す．

第1章　UNIXの基礎

1.1　UNIXとは？

1.1.1　UNIXの基本

UNIXとは，AT&Tのベル研究所で開発されたオペレーティングシステム(OS)である．現在，UNIXは広く普及しており，大型コンピュータからパソコンまでさまざまな機種で利用することができる．UNIXの主な特徴は，**マルチユーザシステム**(フルマルチユーザシステム)である．これは，1台のコンピュータを複数のユーザ(マルチユーザ)で同時に使用できることを意味している．一方，Windowsなどのパソコン OSは1台のマシンを一人のユーザが独占して使用するシステムであり，複数の人間が使用開始時にそれぞれのパスワードを入力して個人個人の環境を構築することができるが，一度に一人しか仕事ができないシングルユーザシステム(擬マルチユーザシステム)になっている．この辺りが，UNIXとパソコンOSとの大きな違いであり，コンピュータシステムを大勢で使っているという意識を常に持ち，コンピュータの資源を無駄に使わないように心掛けたいものである．

UNIXはマルチユーザシステムなので，ユーザ間でセキュリティ問題などが発生しないように厳密に管理されている．この管理を行っているのが**スーパーユーザ**である．スーパーユーザはシステムの保守管理やユーザ管理を行っている．スーパーユーザ以外のユーザを**一般ユーザ**と呼ぶ．

また，UNIXでは，システムの停止を行う場合，**単純に電源を落としてはいけない**．これは，ディスクへの書込みが非同期で行われているからなどの理由による．したがってシステムの停止は，スーパーユーザが行う．

UNIXシステムでは**大文字と小文字は区別されている**．ファイル名やディレクトリ名，コマンド名を使うときやオプションを指定するときには注意が必要である．

UNIXには，先頭が'.'で始まる名前のファイルが存在する．これを**ドットファイル**と呼び，システム管理などに使われる大事なファイルである．通常ドットファイルは隠しファイルになっているので，画面上にリスト表示はされない．

Windowsには，「C:ドライブ」というようにドライブ名が存在していて，ハードディスクドライブやCD-ROMドライブといったハードを管理している．しかし，UNIXにはこういったドライブ名はなく，すべて**ファイルシステム**に組み入れて使用する．

ファイルシステムを持つUNIXでCD-ROMドライブなどの外部記憶装置を使おうと思ったとき，CD-ROMをCD-ROMドライブに入れてすぐ使うことができない．これは，外部記憶装置などをすべてファイルシステムに組み込むためであり，いずれかのディレクトリに接続する必要がある．その作業が**マウント**であり，作業が終わったあとに**アンマウント**という処置をして，ファイルシステムから外部記憶装置を切り放す必要がある．

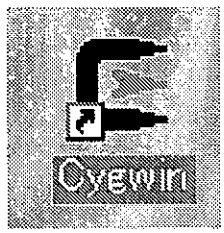
1.2 Windows上でUNIXを使う

1.2.1 Cygwin

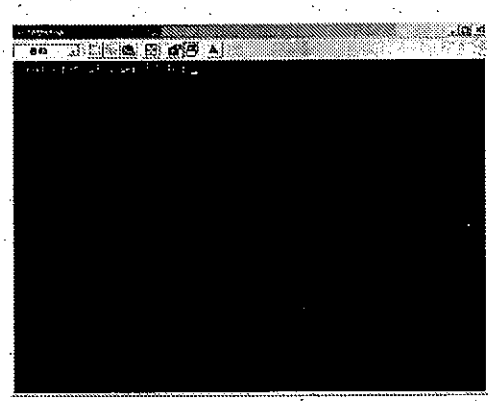
オペレーティングシステム (OS) であるUNIXを利用する場合，UNIXをコンピュータにインストールする必要がある．パーソナルコンピュータ (PC) で起動するUNIX(PC-UNIX) として，Linuxやフリー BSDなどが最近普及しており，PC-UNIXを利用する機会も増えている．しかし，Microsoft Windowsが全盛の昨今，OSとしてPC-UNIXだけを利用するというのは何かと不便である.[1] そこで，異なるOSであるUNIXとWindowsを共存させるために，**デュアルブート (マルチOSシステム)** を採用して，PCをいったん再起動するという手段が一般的にとられているが，この方法だと，利用環境を整えるのも大変煩雑であり，またPC-UNIXがスタンドアロンのOSであるため，UNIXソフトを利用しながらWindowsソフトも利用するといったことができない．そこで，Windows上でUNIX環境を使う方法として，**Cygwin(シグウィン)** というソフトウェアパッケージを利用するという方法がある．Cygwinでは，**cygwin1.dll** というDLL(ダイナミックリンクライブラリ) を介して，UNIXコマンドをWindowsコマンドへ翻訳してWindows上でのUNIX環境を実現している.[2] 一般的なPC-UNIXに匹敵する数のコマンドやソフトウェアがCygwinに移植されており，特別なソフトウェアを使おうとしない限り，全く不自由なくUNIX環境を利用できるはずである．

1.2.2 Cygwinの起動

Windowsを起動すると，デスクトップ上に図 1.1(a) のようなCygwinアイコンがある．そのアイコンをダブルクリックすると，図 1.1(b) のようなCygwinコンソールが起動する．コンソール画面の上部には，



(a) Cygwinアイコン



(b) Cygwinコンソール

図 1.1: Cygwin.

「open00@openlab00:open00(500) $ ■」という文字列が表示される．これには次のような意味がある．

open01	Windowsのログイン名
openlab01	Windowsのホスト名
open01	カレントディレクトリ
500	現在のコマンド履歴番号
$ ■	プロンプト

[1] Officeで作成されたファイルを読み込むことができるフリーのUNIXソフトも開発されている．
[2] Cygwinというソフトウェアがあるわけではなく，Cygwinというプログラムが動いているわけでもない．また，エミュレータを使うよりも快適な環境である．

プロンプトは入力を促す記号で，カーソルが点滅して入力待ちの状態であることを示している．このプロンプトはユーザがカスタマイズすることができる．コンソールのプロンプトから

```
open01@openlab01:open01(500)$ export␣ PS1="$␣ " ⏎
```

と入力すればプロンプトだけが表示されるようになる (この後のテキスト中のプロンプトはすべて$ で記述する)．このプロンプトに対して，⏎ を入力するたびに，UNIX はコマンドを受け付ける．⏎ を押す前ならば，BS を押して誤って入力した文字を１文字ずつ取り消すことができる．

Cygwin を終了するには，

```
$ exit ⏎
```

と入力する．

exit コマンドを使っても Cygwin ターミナルが閉じない場合がある．この場合は Windows の右上の「Window を閉じるボタン」を押す．しかし，この方法だと，シェルの強制終了となるため，コマンドヒストリーが残らなかったり，Windows 上に不要なプロセスが残ってしまうことがあるので，この方法は最終手段としてとどめておくべきである．

1.3 X Window System

X Window System は，Windows などと同じように **GUI(Graphical User Interface)** を提供するソフトウェアで，アイコンなどを操作することでコンピュータを操作，制御する．これに対して，Cygwin コンソールのときのように，文字 (コマンド) でコンピュータを操作，制御することを **CUI(Character User Interface)** という．

前章で起動した Cygwin コンソールで

```
$ startx ⏎
```

と入力すると，画面いっぱいに図 1.2 のような画面が表示される．この Window が X Window System の

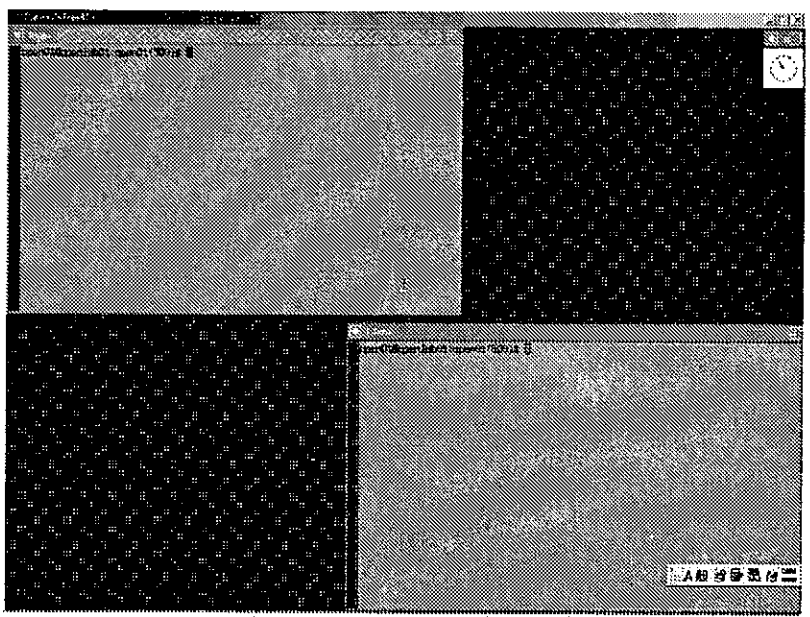


図 1.2: X Window System

ルートウィンドウになり，すべてのウィンドウの親元になる．各種ウィンドウはルートウィンドウ下に表示されている．

図1.2には**XClock**という時計アプリケーションと画面左右に**login**とか**kterm**と名前が付いているウィンドウ(ターミナル)が表示される．このloginやkterm.と名前の付いてるターミナルは，**kterm**と呼ばれる端末エミュレータである．

WindowsからCygwinを起動したときCUIのコンソール(**端末**)が表示され，その中でコマンドを起動することができた(`startx`のこと)．X Window Systemではktermが**端末**の機能を提供するソフトウェアであり，このktermの中でコマンドを起動することができる．なお，loginと表示されているウィンドウは**ログインウィンドウ**と呼ばれ，この端末エミュレータで`exit`コマンドを入力すると，X Window System自体が終了する．

X Window Systemの画面上に「**X**」という記号が表示されているが，これがマウスポインタである．このマウスポインタはマウスを使って移動することができ，端末エミュレータの上にマウスポインタが重なると「**X**」から「**I**」に形状が変わる．マウスポインタが重なっている端末エミュレータがアクティブな端末であり，そこでコマンドの入力などができる．

今回，教育講座で用意したCygwinの環境では，デフォルトで**twm**というウィンドウマネージャが起動している．Cygwinの X Window Systemで利用できるウィンドウマネージャには他にも**fvwm2**や**WindowMaker**などがある．

1.4 リモートへのログイン

1.4.1 利用登録

UNIXは，マルチユーザOSなので，計算機を利用する人間をOSが識別する必要がある．そのため，システム管理者が利用者についてユーザ登録をすると，利用者を区別するUNIX上での名前(**ユーザ名，ログイン名**)が各自に与えられる．また，そのユーザ名を使った人が本人であることを確認するための本人だけが知っている合言葉(**パスワード**)が登録される．このユーザ名を使った計算機の利用許可のことを**アカウント**と呼ぶ.[3]

1.4.2 SSHでのログイン

今回の教育講座では，ローカルマシンであるWindows(Cygwin)から計算機サーバへ**SSH**でログインする．SSHは，ネットワーク経由でコンピュータを遠隔利用する際に使う通信手段`telnet`の通信経路を暗号化することでセキュリティ性を高めた通信手段である．本来ならば，RSA方式やDSA方式で暗号鍵を作成するなどしてよりセキュリティ性を高めた通信を行なうが，今回はplain password(計算機サーバで登録してあるパスワード)を利用してログインする．方法は，プロンプトに対して

```
$ ssh␣ リモートマシンでのログイン名@リモートマシン名↵
```

と入力すると，

```
リモートマシンでのログイン名@リモートマシン名's password:
```

とパスワードの入力を促してくるので，リモートマシンでのパスワードを入力する．もし，初めてのログインの場合，

```
The authenticity of host 'リモートマシン名 (***.***.***.***)' can't be estabished.
RSA key fingerprint is **************.
Are you sure you want to continue connecting (yes/no)?
```

[3] 特別な権限(**ルート権限**)を持つシステム管理者のことを**スーパーユーザ**と呼び，**一般ユーザ**と区別する．何かシステム上のトラブルが発生した場合はスーパーユーザに必ず相談してほしい．

というメッセージが表示されるが，*yes* と入力すればよい．その後，若干のメッセージが表示されパスワードの入力を促されるので，パスワードを入力する．ここで，***.***.***.***はリモートマシンの IP アドレスであり，**********は RSA 鍵の指紋である．

また，リモートマシン上で X を立ち上げる場合は `ssh` にオプション`-X` をつければよい．

1.4.3 パスワード

パスワードは決して他人に知られてはいけない.[4] また，**パスワードは注意して決定しなければならない**．想像可能な文字列，単語 (辞書に掲載されている文字列は問題外)，その逆順やそれらの組み合わせで作るのは避けるべきである．また，定期的にパスワードを変更することでセキュリティを高めることができる．なお，UNIX のパスワードは 8 文字まで解釈され，9 文字目からは認識されない．

パスワード変更には `passwd` コマンドを使う.[5] プロンプトに対して，

```
$ passwd ⏎
Old password: 現パスワード ⏎
New password: 新パスワード ⏎
Retype new password: 新パスワード ⏎
$ ■
```

と実行する．入力されたパスワードは画面に表示されない．また，現在のパスワードを間違えたり，同一の文字列を新パスワードとして 2 回入力しなかった場合は，パスワードは変更されない．

1.4.4 画像ビューア:gv

実習では，シミュレーション結果を画像にして表示する．このとき画像は **Postscript ファイル**という形式で保存される．この Postscript ファイルを表示するためのアプリケーションソフトが gv である．プロンプトに対して，

```
$ gv␣ Postscript ファイル名⏎
```

と入力すればよい．また，画像が横長だった場合 (Landscape)，オプション`-landscape` をつけるか，gv の起動画面の中で Landscape に切れかえればよい．

1.4.5 ログアウト

一通りの作業が終わったら，リモートマシンからログアウトする．プロンプトに対して，

```
$ exit⏎
```

と入力すればよい．

1.4.6 FTP:sftp

ローカルマシン上のファイルをリモートマシンへ転送したり，逆にリモートマシン上のファイルをローカルマシンへ転送したりするためのコマンドが `ftp` である．ただ，`ftp` だけでは通信経路が暗号化されていないので，SSH の ftp である `sftp` を使う．プロンプトに対して，

[4] コンピュータシステムにおける最大のセキュリティホールは，実は人間であるとも言える．パスワードをキーボードの裏に書いておいたり，メモを机の中に忍ばせておいたり，パスワード入力を盗み見られたり...

[5] Network Information Servis (NIS) 環境では `yppasswd` を使う．

```
$ sftp␣ リモートマシンでのログイン名@リモートマシン名↵
```

と入力してパスワードを入力すると,

```
sftp>
```

とプロンプトがsftp>になってリモートマシンとの通信が可能になる. あとは第1.6章で学ぶファイル操作のコマンドがほぼそのまま使うことができる. すなわち, リモートマシン上でのファイル操作として,

ディレクトリやファイルのリスト	ls
ディレクトリの作成と消去	mkdir & rmdir
ディレクトリの移動	cd
カレントディレクトリの表示	pwd
ファイルの消去	rm

を使うことができる. また, ローカルマシン上でのファイル操作では,

ディレクトリやファイルのリスト	!ls
ディレクトリの作成と消去	!mkdir & !rmdir
ディレクトリの移動	lcd
カレントディレクトリの表示	!pwd
ファイルの消去	!rm

のように「!」や「l」を付ければよい.
ファイルの転送を行なうには,

ローカルからリモートへ	put
ローカルからリモートへ(複数のファイル)	mput
リモートからローカルへ	get
リモートからローカルへ(複数のファイル)	mget

をプロンプトに対して入力する.
作業が終わったら,

```
sftp>exit↵
```

と入力すれば, sftpは終了する.

1.5 ファイルシステム

1.5.1 ファイルシステムの構造

UNIXのファイルシステムは**ツリー構造**になっている(図1.3). 図1.3で, 長円で表現された枝別れの部分は**ディレクトリ**と呼ばれる特別なファイルであり, 枝分かれした先に**ファイル**(図1.3では長方形で表現されている)やディレクトリを作成することができる.
この図で, 最上部のディレクトリを**ルートディレクトリ**“ / ”と呼ぶ.

ファイルシステム上のディレクトリ, あるいはファイルを指定するためには**パス名**という文字列を使用する. ルートディレクトリの下にあるlocaluserというディレクトリは, /localuserというパス名で指定することができる.

このように, ルートディレクトリから辿った経路のことを特に**絶対パス名**という. 図1.3にはEという名前をもつファイルが2つあるが, 絶対パスを用いて指定するとそれらを区別することができる. つまり,

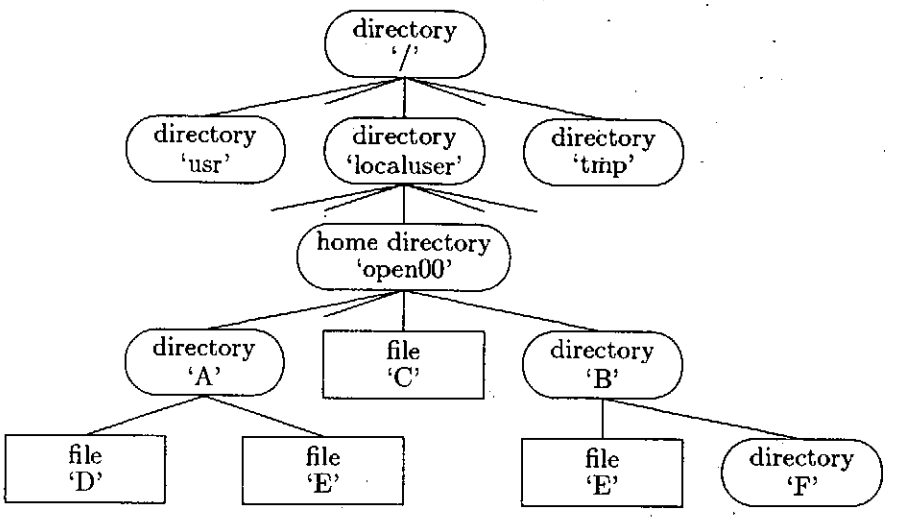


図 1.3: ファイル構造の模式図.

図 1.3 の /localuser/open00/A/E と /localuser/open00/B/E は，同じ E という名前のファイルであるが，絶対パス名で区別することができ，異なるファイルである．なお，同じディレクトリ内に同一の名前のファイルやディレクトリを作ることはできない．

1.5.2 ファイルとディレクトリ

自分で作成した Fortran のプログラムや計算データ，電子メールの内容など，コンピュータのデータはハードディスク上に**ファイル**という単位で保存，管理される．自分で作成するファイル以外にも，あらかじめ UNIX に標準で用意されているファイルも多数存在する．

UNIX のファイルには，基本的に以下の 2 種類のものがある．

- **テキストファイル** – 画面に表示し，人間が読める文書ファイル．通常，テキストエディタを用いて作成する．プログラムのソースファイルや，電子メールの内容など．
- **バイナリファイル** – コンピュータが理解できる形式でデータが保存されているファイル．画面に表示しても人間にとっては意味の無い情報．標準コマンド，またコンパイラを用いて自分で作成したプログラムの実行可能ファイルなど．

これら UNIX のファイルを管理するための機構を提供するのが，**ファイルシステム**である．

前述の通りツリー構造の枝分かれの部分が**ディレクトリ**であり，複数のファイルを格納することができる．そのため，書類を引き出しに整理する感覚と同じように，ファイルをディレクトリに分けて整理することができる．ファイルをたくさん作るようになると，ディレクトリを用いて整理するのが便利で，UNIX も使いやすくなる．また，ディレクトリには，他のディレクトリを格納することもできる．したがって，ディレクトリを利用すると，図 1.3 のような階層的なファイルシステムを構築することができる．

- **カレントディレクトリ**

  現在，作業の対象としているディレクトリのことを**カレントディレクトリ (カレントワーキングディレクトリ)** という．カレントディレクトリから相対的関係でファイルやディレクトリを指定することもできる．たとえば，図 1.3 で，カレントディレクトリが A(/localuser/open00/A) の場合，ファイル D は ./D で指定できる．ここで，“ . ”はカレントディレクトリを表す．この「カレントディレク

トリの下の」という意味の ./ は省略可能である．つまり，./D の代わりに D を，./E の代わりに E を用いることができる．カレントディレクトリが F(/localuser/open00/B/F) の場合，ファイル D は ../../A/D と指定できる．ここで，“ .. ”はカレントディレクトリ F の「ひとつ上の」ディレクトリ (/localuser/open00/B) を表す (注意：ディレクトリの上のディレクトリはただひとつしかない)．次の“ .. ”は，直前に指定されているディレクトリ (/localuser/open00/B) のさらにひとつ上のディレクトリ (/localuser/open00) を表す．このようにカレントディレクトリから辿った経路のことを**相対パス名**と呼ぶ．

- **ホームディレクトリ**[6]

  ログインした直後，カレントディレクトリは**ホームディレクトリ**と呼ばれるディレクトリになっている．ホームディレクトリはシステム構成や利用者ごとに異なり，一般的には，“/home/ユーザ名”のような設定になっている (図 1.3 では，/localuser/open00)．自分のホームディレクトリを指定するのに，“ ~ ”を使うことができる．したがって，図 1.3 でファイル D を ~/A/D と指定することもできる．自分以外の人のホームディレクトリを指定するのに，“ ~ユーザ名 ”を使うことができる．ユーザ名が open00 以外の人は，ファイル D を ~open00/A/D と指定することができる．

1.5.3 UNIXの主なディレクトリ

UNIX に標準で用意されているファイルやディレクトリは多数あり，存在すべきディレクトリやその位置，利用目的などは UNIX のディストリビューションによってほぼ統一されている．以下に UNIX の標準的なファイルシステムのレイアウトを示す．

- **bin**: システムの基本的な動作や修復作業の際に必要になるバイナリが存在するディレクトリ．
- **boot**: カーネルが起動する前に，マシンを起動させるのに使うファイルが存在するディレクトリ．通常は書き換えを行わない．
- **dev**: デバイスファイルが存在するディレクトリ．
- **etc**: 設定ファイルや問題が生じた際に使われるファイルが存在するディレクトリ．
- **home**: ユーザのホームディレクトリ．
- **lib**: ライブラリ，共有ライブラリ，カーネルモジュールが存在するディレクトリ．
- **mnt**: 一時的な用途でマウントするファイルシステムのためのマウントポイント．
- **opt**: システムに追加でインストールされるソフトウェアパッケージが存在するディレクトリ．
- **root**: スーパーユーザ (root) のホームディレクトリ．
- **sbin**: システム管理用コマンドが存在するディレクトリ．
- **tmp**: プログラムが一時ファイルを作成する際の空きスペースとして使われるディレクトリ．
- **usr**: マシン全体で共有されるファイルを格納したリードオンリーのディレクトリ．
- **var**: 頻繁に書き換えられるデータファイルが存在するディレクトリ．

[6] ここでの「ホーム」は基点という意味．ホームベースの「ホーム」と同義で，ホームページの「ホーム」も本来はこの意味である．

1.5.4 アクセスパーミッション:chmod, chown & chgrp(change mode, change owner & change group)

アクセスパーミッションとは，ファイルシステム上のファイルやディレクトリなどの要素に対して，だれが読み取りや，書き込み，実行の権限を持つかを制御することで，3種類の権限設定（**読み取り (r)**，**書き込み (w)**，**実行の権限 (x)**）を3組指定することができる．3組とは，**ファイルの所有者（オーナー）の権限**，**グループの権限**，**その他のユーザの権限**のことである．

- r （3ビットの値：4）ファイルを読み出せるかどうか，ディレクトリを閲覧できるかどうか．
- w （3ビットの値：2）ファイルまたはディレクトリに書き込めるかどうか．
- x （3ビットの値：1）実行できるかどうか．

権限が許可になっている場合，対応する上記の文字がそれぞれの権限を許可していることを示す．逆に不許可になっている場合，ハイフンが表示される．たとえば，オーナーにすべての権限が許可され，グループメンバーには読み出しと実行が許可され，その他のユーザには読み出しだけが許可されているファイルの場合，

```
$ ls␣ -l␣ ファイル↵
rwxr-xr-  open00  openlab  1111  10月14日  13:35  ファイル
```

のように表示される．ここで，「open00」はオーナーを表し，「openlab」はグループを表す．また，「1111」はファイルの大きさをバイト単位で表現している．日付と時間はファイルへの最終アクセス時間である．

アクセスパーミッションの指定方法には2通りの方法がある．一つは9文字表記による指定ともう一つはビット値による指定である．9文字表記による指定の方法は，たとえば，先ほどの権限許可 (rwxr-x-r-) に対して，グループメンバーに書き込み許可を与え，その他のユーザの読み込み許可をはずす場合，

```
$ chmod␣ g+w,o-r␣ ファイル↵
$ ls␣ -l␣ ファイル↵
rwxrwx---  open00  openlab  1111  10月14日  13:35  ファイル
```

というように，「+」や「-」を使って指定する．ビット値を使う方法では，たとえば，先ほどの権限許可 (rwxr-x-r-) をビット値で表現すると，オーナーの組は **rwx=4+2+1=7**，グループユーザの組は **r-x=4+0+1=5**，その他のユーザの組は **r--=4+0+0=4** となるので，これを順に並べると，このアクセスパーミッションのビット値表現は **754** になり．

```
$ chmod␣ 754␣ ファイル↵
$ ls␣ -l␣ ファイル↵
rwxr-xr-  open00  openlab  1111  10月14日  13:35  ファイル
```

というように指定することができる．

また，ファイルの所有者やグループを変更する方法は，プロンプトに対して，

```
$ chown␣ 所有者␣ ファイル↵
$ chgrp␣ グループ␣ ファイル↵
```

と入力すればよい．

1.6 コマンド

1.6.1 オンラインマニュアル:man(manual)

わからないコマンドの操作方法を知るために，厚いマニュアル本などをわざわざ持ってきて開くのはとても億劫である.[7] そんなときのために，UNIXでは，画面上でマニュアルを参照することができるコマンドがある．これを**オンラインマニュアル**といい，Windowsなどでのヘルプに相当する．オンラインマニュアルは，プロンプトに対して，

```
$ man␣ コマンド名↵
```

とタイプすれば良い．また，正しいコマンド名がわからなくても，プロンプトに対して次のようにタイプすれば，キーワードに関連のあるコマンドを表示することができる．

```
$ man␣ -k␣ キーワード↵
```

1.6.2 ファイルの操作

ディレクトリのリスト:ls(list)

ディレクトリの下にどんなファイルやディレクトリが存在しているのかを調べるコマンドがls(list)である．

```
$ ls␣ [ オプション ]␣ [ ディレクトリ名 またはファイル名 ]↵
```

ここで，[ ] で囲った部分は，省略することができる．**ディレクトリ名 または ファイル名** を省略した場合，カレントディレクトリに関する情報が出力される．オプションには次のようなものがある．

- `-l` ファイルに関する詳細な情報(サイズ，日付等)を出力する．
- `-a` 引数なしか，もしくは引数にディレクトリを指定した場合には，通常表示されない(“.”で始まる名前を持つ)ファイルもリストされる．
- `-A` “.”,“..”を除いたすべてのファイルを表示する．
- `-F` ディレクトリの後には /，実行可能ファイルの後には *，シンボリックリンクされているファイルの後には @，ソケットには = をつけて出力する．
- `-R` 再帰的にサブディレクトリの中も表示する．

ディレクトリの作成と消去:mkdir & rmdir(make directory & rmove directory)

ディレクトリを作るには，プロンプトに対し，

```
$ mkdir␣ ディレクトリ名↵
```

とタイプする．

ディレクトリを消すには，プロンプトに対し，

```
$ rmdir␣ ディレクトリ名↵
```

とタイプする．ただし，その下に何も無い空のディレクトリでなければ消せない(1.6.2も参照)．

また，**ディレクトリ名**や**ファイル名**は，特に指定しない限り，カレントディレクトリにあるディレクトリやファイルとみなされる．

[7] もちろん，身近にいる熟達者に尋ねるのもひとつの方法であるが，やはり，まずはじめに自力で解決できるようになるのが一番である．

ディレクトリの移動:cd(change directory)

ディレクトリを移るには，プロンプトに対し

```
$ cd␣/ディレクトリ名↵
```

とタイプする．**ディレクトリ名** を省略した場合には，login した直後のホームディレクトリに戻る．

また，相対パスを用いて，カレントディレクトリの一つ上のディレクトリへ移ることもできる．例えば，図 1.3 で，カレントディレクトリが F の場合，

- ディレクトリ B に移るには，

```
$ cd␣..↵
```

- ディレクトリ A に移るには，

```
$ cd␣../../A↵
```

とタイプすればよい．

カレントディレクトリの表示:pwd(print working directory)

カレントディレクトリの絶対パスを知るためには，プロンプトに対して，

```
$ pwd↵
```

とタイプする．

ファイルの消去:rm(remove)

ファイルを消去するには，プロンプトに対して

```
$ rm␣ファイル名↵
```

とタイプする．

ファイル消去のコマンド rm にはオプションがあり，指定されたファイルを消していいかどうかの判断を，ユーザーに仰ぐようにすることができる．そのためには，プロンプトに対して

```
$ rm␣-i␣ファイル名↵
```

とタイプし，UNIX からのプロンプトに対して，y(es) または n(o) で答えればよい．

また，ディレクトリを消去するコマンドとして rmdir を紹介したが，コマンド rm にオプション-r を付けて，ディレクトリとそのディレクトリ下のファイルを一度に消すことができる．そのためには，プロンプトに対して，

```
$ rm␣-r␣ディレクトリ名↵
```

とタイプする．

ファイルの複写と移動:cp & mv(copy & move)

ファイルの複写または移動をするには，プロンプトに対して

```
$ cp␣ 複写元のファイル名␣ 複写先のファイル名⏎
```

```
$ mv␣ 移動元のファイル名␣ 移動先のファイル名⏎
```

または

```
$ cp␣ 複写元のファイル名␣ 複写先のディレクトリ名⏎
```

```
$ mv␣ 移動元のファイル名␣ 移動先のディレクトリ名⏎
```

とタイプする．前者の場合で、ファイルの移動は，ファイル名の変更と同じことになる．後者の場合では，指定したディレクトリの下に，元のファイル名と同じ名前でファイルが複写または移動される．また，複写または移動先のディレクトリ名を“．”にすると，カレントディレクトリに同じファイル名で複写または移動される．

また，ディレクトリサブツリーごと複写または移動することができる．複写の際には，オプション-rをつける．

```
$ cp␣ -r␣ 複写元のディレクトリ名␣ 複写先のディレクトリ名⏎
```

```
$ mv␣ 移動元のディレクトリ名␣ 移動先のディレクトリ名⏎
```

複写または移動先のディレクトリ名のものがすでにある場合には，そのディレクトリの中に同じ名前で複写または移動される．複写または移動先のディレクトリ名が新しいものであれば，新しい名前として複写または移動される．

空ファイル生成 or ファイルアクセス時刻変更:touch

計算機センターなどでは，「最終アクセス日からある一定期間経過したファイルは自動的に消去する」といった運用を行なっているところは少なくない．消去されたくないファイルがあるときに便利なのが，touchコマンドである．touchコマンドは，空のファイルを作るか，ファイルのアクセス時刻等を変更するコマンドで，次のようにタイプする．

```
$ touch␣ [オプション]␣ [ファイル名]⏎
```

このとき，オプションを何も指定しないと現在の時刻がアクセス時刻となる．オプションには次のようなものがある．

-a	ファイルを最後にアクセスした時刻に更新.
-c	ファイルを作成しない.
-d	指定時間で更新.
-m	ファイルを最後に修正した時刻を更新.
-r	指定ファイル名の時刻で更新.
-t 時刻	時刻 ([[CC]YY]MMDDhhmm:CCYY, 西暦 MM, 月 DD, 日 hh, 時 mm, 分) を指定.

1.6.3 ワイルドカード

ファイル名を指定するのに便利な機能として，ワイルドカードという文字がある．

*	ファイル名とマッチする任意の文字列．
?	ファイル名とマッチする任意の１文字．
[]	[] 内のファイル名とマッチする任意の１文字を表す．例えば，[ace] は，a,c,e のどれか１文字，[a-e] は，a から e までのどれか１文字，[1-5] は，1 から 5 までのどれか１文字を表す．

これらのワイルドカードの対象となるのはディレクトリ内に存在するファイルだけである．これとは別に文字列の展開に有効な表現方法もある．

{}	{} 内の “ , ” で区切られた任意の文字列を表す．

例えば，あるディレクトリ内に t1 , t2 , t3 , t11 , t12 , t13 , sa というファイルがあったとき，以下のようになる．

t*	sa 以外全てのファイルを指定
t?	t1,t2,t3 を指定
*1	t1,t11 を指定
?1	t1 を指定
t[13]	t1,t3 を指定
t[1-3]	t1,t2,t3 を指定
{t1,sa}	t1,sa を指定

ブレース {} はファイル名にマッチングパターンがない時に使うと便利である．例えば，{A,B}/{a.c,f.p} は A/a.c , A/f.p , B/a.c , B/f.p に展開される．ブレース {} は入れ子にしてもよいし，その中にワイルドカードが現れても構わない．

1.6.4 データの入出力

標準入力，標準出力，標準エラー出力

UNIX におけるデータの入出力には，**標準入力**，**標準出力**，**標準エラー出力**という考え方がある．標準入力とは，キーボードやパイプライン (後述 1.6.4) の直前のコマンドから入力を与えることで，標準出力とは，画面上やパイプラインの次のコマンドへ出力を与えることを意味している．標準エラー出力はエラー用で，通常は画面上に出力される．

また，Fortran において，特に指定しない限り，

read(5,···),read(*,···)	は標準入力から入力を受け，
write(6,···),write(*,···)	は標準出力に出力し，
write(0,···),write(*,···)	は標準エラー出力に出力する．

リダイレクション

通常割り当てられている入出力先（キーボードや画面上）から別の入出力先（例えばファイル）に切り替えたいとき，標準入力先，標準出力先を切り替え指定することができる．これを**リダイレクション**と呼び，コマンドに対して次のような記号を使うことでリダイレクションができる．

< ファイル名	標準入力をそのファイルから取る.
コマンド|コマンド	左側のコマンドの標準出力を右側のコマンドの標準入力とする.
コマンド|&コマンド	左側のコマンドの標準出力と標準エラー出力を右側のコマンドの標準入力とする.
> ファイル名	標準出力をファイルの頭から書き込む. 通常, そのファイルが既に存在する場合, 書き込まない設定になっている.
>> ファイル名	標準出力を既に存在するファイルの末尾へ追加する.
>! ファイル名	標準出力をそのファイルの頭から書き換える. そのファイルが既に存在する場合, 既存の内容を消してから書き込む.
>& ファイル名	標準出力と標準エラー出力の両方をその新しいファイルへ (ファイルの頭から) 書き込む. 通常, そのファイルが既に存在する場合, 書き込まない設定になっている.
>>& ファイル名	標準出力と標準エラー出力の両方をその既存のファイルの末尾へ追加する.
>!& ファイル名	標準出力と標準エラー出力の両方をその既存のファイルへ (ファイルの頭から) 書き換える. そのファイルが既に存在する場合, 既存の内容を消してから書き込む.

次の例は, 通常画面に出力する ls コマンドをファイルへの出力にリダイレクトしている.

```
$ ls␣ > ␣ file1↵
```

このように, キーボードから手で入力していたパラメータをファイルから入力したり, ディスプレイ上に出していた出力をファイルに格納したりすることができる.

ファイル内容の出力:cat(concatenate)

指定したファイルの内容を画面上に表示するには, プロンプトに対して,

```
$ cat␣ ファイル名↵
```

とタイプする. cat コマンドは複数のファイルを結合して表示することもできるので, ファイルを連結するのに便利である. しかし, ファイルの中身を一気に表示するので, ファイルが大きいと内容が一画面に収まらずに, 文字が流れてしまう (そんなときには, 1.6.4 の more や less を使う).

また, 次のようにすると行番号付きで表示する.

```
$ cat␣ -n␣ ファイル名↵
```

ファイル内容の出力 (ビューア):more & less

一画面には収まりきらないような大きなファイルを一画面ずつ表示することができるコマンドが more と less である.

```
$ more␣ ファイル名↵
```

または,

```
$ less␣ ファイル名↵
```

ファイルの内容を出力する．ディスプレイの１画面分を出力すると，最下行に“：”と出力して一旦出力を停止する．そこで以下のような文字を入力すると次のように動く．

Space	次の１画面分を出力する．
⏎	次の１行を出力する．
b	前の１画面分を出力しなおす．
k	１行分出力画面を戻す．
q	出力はそこで打ち切られる．
h	サブコマンドの説明が表示される (help 機構)．

但し，標準的な more には，前の画面を表示する機能 b がない．

ファイル名を指定しない場合，more , less コマンドは標準入力より入力を受け，標準出力に出力する．そこで，あるコマンドの結果が１画面分を越えるような場合，

```
$コマンド| more⏎
```

または

```
$コマンド| less⏎
```

とすると便利である．

ファイルの先頭・終端部分の表示:head & tail

ファイルの先頭部分や終端部分をざっと表示させるときに使うコマンドが，それぞれ，head と tail コマンドである．

```
$ head␣ ファイル名⏎
$ tail␣ ファイル名⏎
```

このようにタイプすると，head ではファイルの先頭１０行を表示し，tail ではファイルの最後１０行を表示する．また任意の行数を表示するには，次のようにすればよい．

head␣ -*n*␣ ファイル名⏎	ファイルの先頭から *n* 行目までを表示
tail␣ -*n*␣ ファイル名⏎	ファイルの最後から *n* 行目までを表示
tail␣ -*n*␣ ファイル名⏎	ファイルの *n* 行目からを表示

これらを組み合わせることで，ファイルの中間部分を表示することができる．

head␣ -*9*␣ ファイル名 | tail␣ -4⏎	ファイルの６～９行目を表示
tail␣ +*6*␣ ファイル名 | head␣ -4⏎	ファイルの６～９行目を表示
cat␣ -*n*␣ ファイル名 | tail␣ +*6* | head␣ -4⏎	ファイルの６～９行目を行番号付きで表示

プリンタへの出力

PostScript ファイルを印刷するには，プロンプトに対し

```
$ lpr␣ -P プリンタ名 ␣ / オプション / ␣ PostScript ファイル名⏎
```

をタイプする．但し，**-P プリンタ名** を省略した場合にはシェルの環境変数 PRINTER で指定されたプリンタから出力される．この作業を**プリンタキュー**を送ると言う場合がある．

この送ったプリンタキューを表示するには，プロンプトに対し

```
$ lpq␣ -P プリンタ名 ⏎
```

をタイプする．但し，**-P プリンタ名** を省略した場合にはシェルの環境変数 PRINTER で指定されたプリンタについての情報を表示する．

プリンタキューを取消すには，まず，**lpq** コマンドでプリントジョブ番号を確認し，プロンプトに対し

```
$ lprm␣ -P プリンタ名␣ プリントジョブ番号⏎
```

をタイプする．但し，**-P プリンタ名** を省略した場合にはシェルの環境変数 PRINTER で指定されたプリンタについて取消される．

ここまでは Postscript ファイルをプリンタで印刷する方法について述べたが，プログラムやファイル，出力結果などのテキストファイルを Postscript プリンタに印刷するためには，テキストファイルを Postscript ファイルに変換してから印刷する必要がある．そのために，**a2ps** コマンドなどを用いる．プロンプトに対し

```
$ a2ps␣ -P プリンタ名␣ ファイル名⏎
```

とタイプする．但し，**-P プリンタ名** を省略した場合にはシェルの環境変数 PRINTER で指定されたプリンタから出力される．

1.6.5 正規表現

正規表現とは，文字列の集合を一定の規則にしたがって表す手法である．正規表現は，^ , $, [] , . , * , \ の 6 種類の記号で制御される．これらの特殊文字や通常の文字列を組み合わせることで，表したい文字列の集まりを記述することができ，後述する文字の抽出コマンドや文字列の置換コマンド (1.6.6 を参照)，**vi** コマンド中の **ex** モード (1.8.5 を参照) で利用することができる．

^	行頭.
$	行末.
.	任意の 1 文字.
[]	[ ] でくくられた文字の集まりの内の 1 文字.
*	直前表現の 0 回以上の繰り返し.
\	直後にくる正規表現での特殊文字を単なる文字として取り出す.
\(\)	グループ化 (置換文字列では，\数字).

マッチするパターンの例

今，例として以下のような文章があるとする．

coffee
cake
cocoa
cookie
tee
$10

次のような検索文字列を指定するとマッチするパターンは次表のようになる．

検索文字列	マッチするパターン	説明
^c	coffee, cake, cocoa, cookie	行頭に c がある行.
ee*	coffee, cake, cookie, tee	e が 1 文字以上含まれる行.
ee$	coffee, tee	行末が ee の行.
\$	$10	$ が含まれている行.

置換の例

正規表現を使った置換の例をいくつか挙げる (vi コマンドの ex モードを例にしている).

1. 1 行目から終りまでに出てくるすべての “dose” を “does” に置換する.

```
:1,$s/dose/does/g
```

2. 10 行目から 20 行目の中で, 最初に出てきた “Windows95” だけを “Macintosh” に置換する.

```
:10,20s/Windows95/Macintosh/
```

3. カーソル行に出てくる “c” で始まり, “k” で終わる 4 文字の文字列すべてを “abcd” に置換する.

```
:s/c..k/abcd/g
```

4. 1 行目らか 50 行目までに出てくる各行の “c” 以降の文字列を “*” に置換する.

```
:1,50s/c.*/\*/g
```

5. 20 行目以降に出てくる “Th” で始まり, “s.” で終わる行すべてを “(comment out)” に置換する.

```
:20,$s/^Th.*s\.$/(comment␣out)/g
```

6. 文章中に出てくるすべての “bluesky” を “skyblue” に置換する.

```
:%s/\(blue\)\(sky\)/\2\1/g
```

7. カーソル行以降に出てくるすべての “There were pens.” を “There are pens.” に置換する.

```
:.,$s/\(There␣\)were\(␣pens\.\)/\1are\2/g
```

8. 1 行目からカーソル行に出てくる “12” で始まり, 途中に “67” があり, 最後が “90” である単語すべての途中の “67” の部分を “123” に置換する.

```
:1,.s/\(12.*\)67\(.*90\)/\1123\2/g
```

1.6.6 情報の抜き出し

文字の抽出:grep(global regular expression print)

grepコマンドは，テキストファイルを1行ずつ見ていき，指定された文字列がある行を表示するコマンドである．

```
$ grep␣ /オプション/␣ "文字列"␣ ファイル名⏎
```

複数のテキストファイルについて調べることや標準入力に対して調べることも可能である．オプションには次のようなものがある．

- -n 指定した文字列を含む行番号も表示．
- -v 指定した文字列を含まない行を抽出．
- -i 指定した文字列とファイルの両方で英字の大文字と小文字を同一視．

文字列の置換:sed(stream editor)

sedコマンドは，テキストファイルを検索したり，正規表現などを使って文字列を置換することができるコマンドである．

```
$ sed␣ 's/検索文字列/置換文字列/'␣ ファイル名⏎
```

ここで，置換文字列を省略すると(//)，検索文字列を削除することができる．もちろん，標準入力に対して調べたり，結果を標準出力に出力して次のコマンドへ引き渡したり，ファイルに出力することも可能である．また，上記のコマンドでは，各行で最初に見つけた検索文字列だけを置換するが，ファイル内すべての検索文字列を置換文字列に置換する場合は，

```
$ sed␣ 's/検索文字列/置換文字列/g'␣ ファイル名⏎
```

のようにタイプする(置換の例は1.6.5を参照)．さらに複雑な作業をこなすことができるgwakというコマンドもある．

ファイルの比較:diff(difference)

diffコマンドは，2つのファイルを行単位で比較して，違いがある行を表示するコマンドである．ファイル名に「-」を指定すれば，標準入力とファイルを比較することができる．

```
$ diff␣ ファイル名1␣ ファイル名2⏎
```

ファイルのバイト数・単語数・行数の表示:wc(word counter)

wcコマンドは，ファイルのバイト数，単語数，行数を表示するコマンドである．

```
$ wc␣ /オプション/␣ ファイル名⏎
```

オプションには次のようなものがある．

- -c バイト数をカウント．
- -l 行数をカウント．
- -w 単語数をカウント．

1.6.7 文字列の並べ替え:sort

sort コマンドは，指定のフィールド (カラム) を対象にしてファイルを行単位で並べ替え (ソート) したり，ファイルがソートされているか調べたり，複数のファイルをソートすることができる．ただし，複数のファイルをソートするときは各ファイルがすでにソートされている必要があり，複数のまとまった結果が出力される．

```
$ sort␣ [オプション]␣ ファイル名...↵
```

フィールドを区切る記号は，デフォルトでは空白が指定されている．したがって，Excel やデータベースソフトで作成したデータもタブ区切りのテキストデータにしておけば簡単にソートができる．もし，区切り記号が空白やタブではなく「:」などのデータファイルの場合，「-t:」というようにオプションをつければよい． フィールドを指定するオプションは「-k」である．オプションには次のようなものがある．

-c	チェックモードで動作.
-m	マージモードで動作.
-k, --key=フィールド位置 1[, フィールド位置 2]	フィールド位置 1 から フィールド位置 2 をソートキーに指定.
-d	アルファベット，数字，空白だけを辞書順にソート.
-f	大文字と小文字を区別しない.
-g	数値としてソート.
-i	ASCII コード以外を無視.
-r	逆順にソート.
-t	区切り文字の指定 (デフォルトは空白).
-o	指定ファイルに結果を出力.

1.6.8 ファイルの圧縮解凍

コンパイル時に発生するオブジェクトファイルなど必要としないファイルなどは消去 (rm) してしまえばよいが，今は必要としないが保存しておきたいソースファイルなどをそのままにしておいては資源の無駄である．ファイルを消去せずにファイルの容量を小さく圧縮するコマンドが comress や gzip，bzip2 である．(圧縮率は，comress，gzip，bzip2 の順で高くなる)

compress & uncompress

compress を使ってファイルを圧縮するときは，

```
$ compress␣ 圧縮するファイル名↵
```

とタイプする．compress コマンドで圧縮すると，圧縮されたファイルには.Z という拡張子が付く．また，compress コマンドで圧縮されたファイルと解凍するには，

```
$ uncompress␣ 圧縮するファイル名.Z↵
```

とタイプすればよい．

gzip & gunzip

gzip を使ってファイルを圧縮するときは，

```
$ gzip ␣ 圧縮するファイル名 ⏎
```

とタイプする．gzipコマンドで圧縮すると，圧縮されたファイルには.gz という拡張子が付く．また，gzipコマンドで圧縮されたファイルと解凍するには，

```
$ gunzip ␣ 圧縮するファイル名.gz ⏎
```

または，

```
$ gzip ␣ -d ␣ 圧縮するファイル名.gz ⏎
```

とタイプすればよい．

bzip2 & bunzip2

bzip2を使ってファイルを圧縮するときは，

```
$ bzip2 ␣ 圧縮するファイル名 ⏎
```

とタイプする．bzip2コマンドで圧縮すると，圧縮されたファイルには.bz2 という拡張子が付く．また，bzip2コマンドで圧縮されたファイルと解凍するには，

```
$ bunzip2 ␣ 圧縮するファイル名.bz2 ⏎
```

または，

```
$ bzip2 ␣ -d ␣ 圧縮するファイル名.bz2 ⏎
```

とタイプすればよい．ファイルを解凍すると，デフォルトでは元のファイルが削除されてしまうので，元のファイルを残しておくときは-k オプションをつける．

複数のファイルをまとめる:tar(tape archive)

tarコマンドは，ディレクトリや複数のファイルを1つのファイルにまとめる（アーカイブする）ためのコマンドである．tarファイルには，ファイル名の最後に拡張子 .tar をつける．新しくアーカイブを作成するときは，

```
$ tar ␣ cvf ␣ ファイル名.tar ␣ まとめるファイル名やディレクトリ名 ⏎
```

とタイプする．まとめるファイル名やディレクトリ名は，「␣」で区切り，複数書くことができる．

アーカイブからファイルを解凍するときは，

```
$ tar ␣ xvf ␣ ファイル名.tar ␣ [ 展開するファイル名やディレクトリ名 ] ⏎
```

とタイプする．展開するファイル名やディレクトリ名 を省略した場合はすべて展開される．

アーカイブに含まれるファイルを一覧表示するには，

```
$ tar ␣ tvf ␣ ファイル名.tar ⏎
```

とタイプする．

アーカイブの最後にファイルを追加するには，

```
$ tar ␣ rvf ␣ ファイル名.tar ␣ [ 追加するファイル名やディレクトリ名 ] ⏎
```

とタイプする.

また，先述の圧縮コマンド compress や gzip，bzip2 で圧縮されたアーカイブを tar コマンドから直接解凍することもできる．-Z オプションで compress コマンド，-z オプションで gzip コマンド，-j オプションで bzip2 コマンドをそれぞれ呼び出すことができる.

```
$ tar ␣ xvfZ ␣ ファイル名.tar.Z ␣ [展開するファイル名やディレクトリ名] ↵
$ tar ␣ xvfz ␣ ファイル名.tar.gz ␣ [展開するファイル名やディレクトリ名] ↵
$ tar ␣ xvfj ␣ ファイル名.tar.bz2 ␣ [展開するファイル名やディレクトリ名] ↵
```

uuencode & uudecode

巨大なファイルを直接電子メールなどで送るのはネットワークにとって大変な負担となってしまう．そこで，ここまでの述べてきた圧縮系コマンドでファイルの容量を小さくして送るのは非常に有効である．しかし，圧縮系ファイルで圧縮されたファイルはバイナリファイルなので，直接電子メールなどで送ることができない．そこで，バイナリファイルをテキストファイルに変換する（エンコードする）ためのコマンド uuencode を利用する.

```
$ uuencode ␣ エンコードするファイル ␣ エンコード名 ␣ > ␣ ファイル名.uu ↵
```

引数**エンコード名**は，uuencode 形式中に必要な名前で，データ復元時に使われる．こうして作られたファイルを電子メールの本文を書くときにエディタで挿入して送る.

uuencode 形式で変換されたデータを元に戻すには，

```
$ uudecode ␣ ファイル名.uu ↵
```

とすれば，**ファイル名**.*uu* に含まれていた uuencode 形式のファイルが元に戻され，先ほどの**エンコード名**で指定したファイル名で復元される．**ファイル名**.*uu* の**ファイル名**ではなく，**エンコード名**が使われることに注意したい.

1.6.9 その他のコマンド

端末のクリア:clear

clear コマンドは端末スクリーンをクリアするコマンドである．端末によっては Ctrl + l でクリアできる場合もある.

```
$ clear ↵
```

ファイルの検索:find

find コマンドは，指定したディレクトリ階層下のファイルを検索するコマンドである．ディレクトリの指定には，/usr/ ではなく，/usr か /usr/. を指定しなければならない.

```
$ find ␣ [パス] ␣ [オプション] ␣ [評価式] ↵
```

[評価式] で検索したい名前や時間，所有ユーザ名などを指定する．例えば，libX11.a というファイルを検索したい場合，

```
$ find ␣ / ␣ -name libX11.a ␣ -print ↵
```

などとタイプする.

実行ファイルの検索:which

whichコマンドは，PATH環境変数に指定されたディレクトリから指定したコマンドを検索するコマンドである．デフォルトでは一番初めに見つけたパスとコマンドを表示し，-aオプションを使うと見つかったコマンドとパスをすべて表示する．使いたいコマンドがPATH環境変数に指定されたディレクトリ内にあるかどうかを確認することにも使える（これをPATHが通っているかどうかという）．

```
$ which ␣ [-a] ␣ /コマンド名/⏎
```

ディスクの使用状況の表示:df & du(disk free & disk usage)

dfコマンドは，ディスクの使用/未使用領域を表示するコマンドである．

```
$ df ␣ /ディレクトリ名/⏎
```

duコマンドは，ファイルやディレクトリの利用サイズを集計して表示するコマンドである．ディレクトリ毎の集計を見たい場合は-sオプションを使う．

```
$ df ␣ /オプション/ ␣ /ディレクトリ名/⏎
```

漢字コードの変換:nkf(network kanji filter)

日本語をUNIXで取り扱う場合，漢字コードというものに注意する必要がある．UNIXで日本語表現を行なうとき，通常日本語EUCコードを使用するが，Windowsではシフト JISコードを使う．また，メールで日本語で文章を送りたい場合，JISコードを使うのが通例となっている．このように異なるシステム間でファイルをやり取りする場合に，漢字コードを変換するコマンドがnkfコマンドである．JISコードへ変換する場合はオプション-j，シフト JISコードへ変換する場合はオプション-s，EUCコードへ変換する場合はオプション-eを使う．

```
$ nkf ␣ -j ␣ /ファイル名 1/ ␣ > ␣ /ファイル名 2/⏎
$ nkf ␣ -s ␣ /ファイル名 1/ ␣ > ␣ /ファイル名 2/⏎
$ nkf ␣ -e ␣ /ファイル名 1/ ␣ > ␣ /ファイル名 2/⏎
```

マウント:mount

WindowsではCDROMドライブなど外部記憶装置をドライブ名で利用しているが，UNIXではディレクトリ構造を採用しているため，これらのドライブはいずれかのディレクトリに接続してシステム内に組み込む必要がある．もしCDROMドライブのドライブ名がWindowsでEだとすると，Cygwinでは「/cygdrive/e」と指定される．そこで，例えば，マウント先を/cdromとする場合，mkdirコマンドでディレクトリ/cdromを作成し，mountコマンドでマウントする．

```
$ mkdir ␣ /cdrom⏎
$ mount ␣ /cygdrive/e ␣ /cdrom⏎
```

作業が終わったら，umountコマンドでアンマウントする．

```
$ umount ␣ /cdrom⏎
```

ちなみに，Linuxでは次のようにコマンドを入力するだけで，CDROMドライブが/mnt/cdromにマウントされる．

```
$ mount ␣ /mnt/cdrom⏎
```

1.7 シェル (bash)

シェルは OS の中心であるカーネル[8] とユーザを橋渡しするコマンドで，ユーザの指示をカーネルに伝える役割がある．

シェルをうまく使いこなすことができるようになれば，CUI 環境における作業効率を格段に上げることができるようになる．すなわち，シェルには，作業を軽減してくれる機能が多数備わっているからである (**エリアス** (省入力機構) や**シェルスクリプト**)．

ちなみに，Cygwin のデフォルトで採用されているシェルは **bash** シェルであるが，そのほかにも **sh** や **ksh** などの **sh 系**，**ch** や **tcsh** などの **csh 系**がある．

1.7.1 シェルの起動

第 1.2 章での Cygwin の起動や第 1.3 章での X Window System の起動が済んでいれば，Cygwin ターミナルや端末エミュレータ上でシェルが起動しているはずである．

1.7.2 コマンドの補完と編集

長いコマンドラインや憶えにくいコマンドを入力する時に便利な機能としてコマンド入力の**補完機能**がある．この機能は，コマンド名やファイル名 (ディレクトリ名も含む) に対して機能する．例えば，

```
$ cat ␣ myfile1.txt ⏎
```

と入力するとき，最初の文字列「diff」を入力する途中で Tab を押すと，コマンド名が補完される．また，2 番目の文字列「*myfile1.txt*」の入力途中で Tab を押すと，ファイル名が補完される．また，補完候補が複数あってその入力文字列だけでは特定できない場合，もう一度 Tab を押すと，その全候補が表示される．

コマンドラインで入力した文字列は ⏎ を押す前であれば，編集することができる．編集方法は以下のとおりである．

Ctrl + b , ←	左に 1 文字移動
Ctrl + f , →	右に 1 文字移動
↑ , ↓	ヒストリ機能
Alt + b , Esc b	左に 1 単語移動
Alt + f , Esc f	右に 1 単語移動
Ctrl + A	先頭に移動
Ctrl + E	行末に移動
BS	カーソル位置の左文字を消去
Ctrl + D	カーソル位置の文字を消去
Ctrl + U	カーソル位置の左文字までを全消去
Ctrl + K	カーソル位置以降の文字を全消去
Alt + D , Esc K	カーソル位置以降の 1 単語を消去

また，マウスを使ってコマンドラインをコピー&ペーストすることもできる．画面上に表示されている必要な文字列をマウスの左ボタンドラッグで反転表示させ，マウスの中ボタン (2 つボタンの場合は左ボタンと右ボタンを同時に) をクリックすると，反転させた文字列が現在にコマンドラインのカーする位置にコピーされる．

[8] Cygwin でカーネルというと，NT カーネルや Windows9x カーネルを示す

入力するコマンドラインが長くなったとき，そのまま入力し続けることもできるが，複数行に分割することもできる．例えば，

```
$ cat ␣ myfile1.txt ␣ myfile2.txt ␣ myfile3.txt ⏎
```

は，

```
$ cat ␣ myfile1.txt ␣ myfile2.txt ␣ \⏎
> myfile3.txt⏎
```

と同じコマンドラインになる．ただし，

```
$ cat ␣ myfile1.txt ␣ myfile2.txt\⏎
> myfile3.txt⏎
```

と入力すると，

```
$ cat ␣ myfile1.txt ␣ myfile2.txtmyfile3.txt⏎
```

という意味になってしまうので注意したい．

1.7.3 コマンドヒストリ

コマンドヒストリ（操作履歴）とは，一度実行したコマンドを記憶しておく機能である．
コマンドヒストリを表示するには，

```
$ history ⏎
```

と入力すれば，これまでに実行したコマンドがヒストリ番号とともに表示される．
一度実行したコマンドを表示して再実行するには，「↑」（上矢印）キー（あるいは Ctrl + P ）を押すとひとつ過去のコマンドへ遡ることができるので，目的のコマンドを探して実行すればよい．また，「→」（右矢印）キーや「←」（左矢印）キーでカーソルを移動することができるので，「↑」キーでコマンドを表示した後，カーソルを移動してコマンドを編集することもできる．新しいヒストリに戻るには「↓」（下矢印）キー（あるいは Ctrl + n ）を押せばよい．
ヒストリ番号を指定して，該当するコマンドを実行することができる．指定の方法は，プロンプトに対して，

!n	n番目のコマンドの実行
!-n	n個前のコマンドの実行
!!	前回のコマンドの実行

である．
先頭文字列の一致するコマンドを再実行するには，**!コマンドの先頭文字列**　をプロンプトに入力すればよい．

```
$ !コマンドの先頭文字列 ⏎
```

指定文字列を含むコマンドを再実行するには，**!?コマンドの先頭文字列?**　をプロンプトに入力すればよい．

```
$ !?コマンドの先頭文字列? ⏎
```

コマンドを部分置換して再実行するには，**^文字列１^文字列２^**　をプロンプトに入力すればよい．

```
$ ^文字列１^文字列２^ ⏎
```

!コマンドの先頭文字列や**!?コマンドの先頭文字列?**，**^文字列１^文字列２^**では，検索したコマンドがいきなり実行されてしまう．これを一度確認して実行したり，編集して実行したい場合には，

コマンドの最後に":p を付けて

実行すればよい．これはコマンドのエコー表示だけをする機能で，結果として，このとき入力したコマンドが「前回のコマンド」に設定されることになる．

また，ヒストリの中からコマンドを検索する機能もある (**reverse-i-search**)．プロンプトで，Ctrl + r を押すと

```
$ (reverse-i-search) `' :
```

と表示されるので，検索したいコマンドの文字を入力すると，１文字入力するたびに，そこまでの文字列と一致するコマンドが表示される．もう一度，Ctrl + r を押すと，次に一致するコマンドが表示される．希望するコマンドが表示されたら，次の操作をして，そのまま実行，あるいは編集実行することができる．

⏎	実行
Esc	検索終了して編集可能
→，←	検索終了して編集可能になりカーソルがひとつ移動
↑，↓	現在のコマンド位置を基点にして，前後にヒストリ移動

一つ前に実行した引数を再利用することもできる．例えば，

```
$ cat ␣ myfile.txt⏎
```

と実行した後，

```
$ vi ␣ !$⏎
```

と入力すると，これは，

```
$ vi ␣ myfile.txt⏎
```

と同等になる．

1.7.4 ディレクトリ名の保存

頻繁にディレクトリを移動すると，いちいちディレクトリ名をコマンドラインに入力するのが億劫になってくる．こんなとき，便利なのが pushd と popd コマンドである．pushd はディレクトリ名のパスを保存し，popd は保存したディレクトリ名のパスを引き出す機能を持っている．たとえば，

```
$ pushd ␣ .⏎
```

と入力すると，現在のディレクトリ名を保存し，

```
$ pushd ␣ /usr/local/bin⏎
```

と入力すると，ディレクトリ名を保存するとともに*/usr/local/bin* ディレクトリに移動することができる．

保存したディレクトリは

```
$ popd⏎
```

と入力すれば，最後に pushd したディレクトリへ移動することができる．このとき，popd すると最後に pushd で保存されたディレクトリ名は消去される．つまり，pushd を実行するたびにディレクトリ名は積み上げられるように保存され，popd を実行するたびに一番上に積み上げられたディレクトリ名が消去されていくことになる．

1.7.5 エイリアス

頻繁に使うコマンドラインをより少ないキー入力で実行できるようにする機能が**エイリアス(別名定義)**である．よく使うオプションをあらかじめ組み込んでおいたり，コマンド名の短縮形を定義しておくことなどができる．例えば，

```
$ alias ␣ rm="rm ␣ -i"⏎
```

と入力すると，次回からrmをコマンド入力すれば，rm ␣ -iと同じ意味になる．

また，

```
$ alias ␣ pd="popd"⏎
```

と入力すれば，次回からはpopdと入力する代わりにpdと入力すればよいことになる．

aliasコマンドを使ってコマンドラインのエイリアスを設定した場合，exitコマンドでシェルを停止すると，設定したエイリアスは無効になってしまう．そこで，シェルを起動するたびにエイリアスを有効にするには，後ほど説明するシェルの設定ファイル.bash_profileに

```
alias ␣ rm="rm ␣ -i"
```

を書き込んでおけばよい．

ただし，エイリアスを定義する場合，定義したコマンド名が既存のコマンド名と重複していないかどうかに注意すべきである．既存のコマンド名があるのにエイリアスを定義すると，エイリアスのほうが有効になって既存のコマンドが使えなくなるからである．エイリアスを定義するときにはwhichコマンドやfindコマンドを使って既存のコマンド名として存在していないかどうか調べるとよい．

1.7.6 環境設定

ここまで見てきたようにUNIXには様々な機能があるが，それらをより使いやすくより自分の環境に合わせていく(**カスタマイズ**する)ことが必要になってくる．Windows GUIなどはとっつきやすく初心者には使いやすい環境であるが，使っていくうちに操作手順などを使い勝手のよいように変更したくなっても変更することが難しい．しかし，UNIXではそういった変更，カスタマイズが容易にできる．ここでは，カスタマイズに必要な環境設定ファイルについて簡単に紹介する．

bashの環境設定ファイル

bashの環境を設定するファイルには以下のものがある．

/etc/profile	bashユーザすべてに作用する．システム管理者がデフォルトの環境を設定する．
~/.bash_profile	ユーザが設定できるファイル．
~/.bash_login	ユーザが設定できるファイル．
~/.profile	ユーザが設定できるファイル．
~/.bash_logout	ユーザが設定できるファイル．
~/.bashrc	ユーザが設定できるファイル．
~/.input	ユーザが設定できるファイル．

これらのファイルはbash起動時に次のような順番で読み込まれる．

1. /etc/profileが存在すれば，読み込んで実行．

2. ~/.bash_profile, ~/.bash_login, ~/.profile の順番に探索し，最初に見つかったファイルだけを読み込んで実行.

3. 終了するときに~/.bash_logout があれば読み込んで実行

~/.bashrc は，ログインシェルではなく，かつ対話的シェルとして bash が起動されるときに読み込まれるファイルで，~/.input は bash 入力時の設定ファイルで，起動するたびに読み込まれる.

したがって，これらのファイルがいつ読み込まれか，どんな順番で読み込まれるかを考えて，自分の環境設定 (エイリアスなど) をファイルに書き込めばよい.

1.8 エディタ (vi)

Linux などでは標準のエディタとして Mule(もしくは他の Emacs 系エディタ) がよく使われている．しかし，Emacs は標準コマンドではなく，環境によっては「入手」して Emacs なしでファイルをエディットする必要が生じるかもしれない．そんなときに使うのが，ほぼすべての UNIX システムに標準で用意されている vi エディタである．また，UNIX システムにトラブルが発生して最小システムでしか動作しないときなどでも vi ならコンソールから実行できる．そこで，ここでは，vi の操作方法について述べる.

1.8.1 vi の起動

vi を起動するには，プロンプトに対して，

```
$ vi ␣ ファイル名 ⏎
```

とタイプする．**ファイル名**は，新しく作ろうとするファイルや，編集しようとするファイルの名前である.

vi が起動すると，編集するファイルの内容が画面に表示される.

```
S imulation Science Open Seminar.
Simulation Research Tutorial Course.
```

ファイルの内容は，文字の集まりで，いくつかの行に分かれている．新しいファイルを作ろうとしたときには，すべての行が空で表示される.

また，ファイルの内容が画面に表示されたとき，画面の左上隅の一文字が点滅する四角の中に表示される．この四角が**カーソル**と呼ばれ，文字を書き入れたり，文字を消したりする位置を指定するのに用いる.

1.8.2 vi のモード

vi を操る上で，**モード**という概念が重要である．vi のモードには次の 3 つがある．サブコマンドを受け付ける**コマンドモード**と，文字列を書き込む**テキスト入力モード**，vi 実行中に ex エディタの操作を行う **ex モード**である (図 1.4).

コマンドモードは，カーソルの移動や文字の挿入，行の挿入，文字削除，行削除などを行うことができる状態である．テキスト入力モードでは，タイプした文字が画面上に書き込まれ，BS で入力した文字を取り消すことができる．ex モードは，ファイルへの保存やファイルの読込み，行指定や行に対する編集を行うことができる状態である.

UNIX のプロンプト状態とこれら 3 つのモードを切り替える方法が図 1.4 の矢印で示されている.

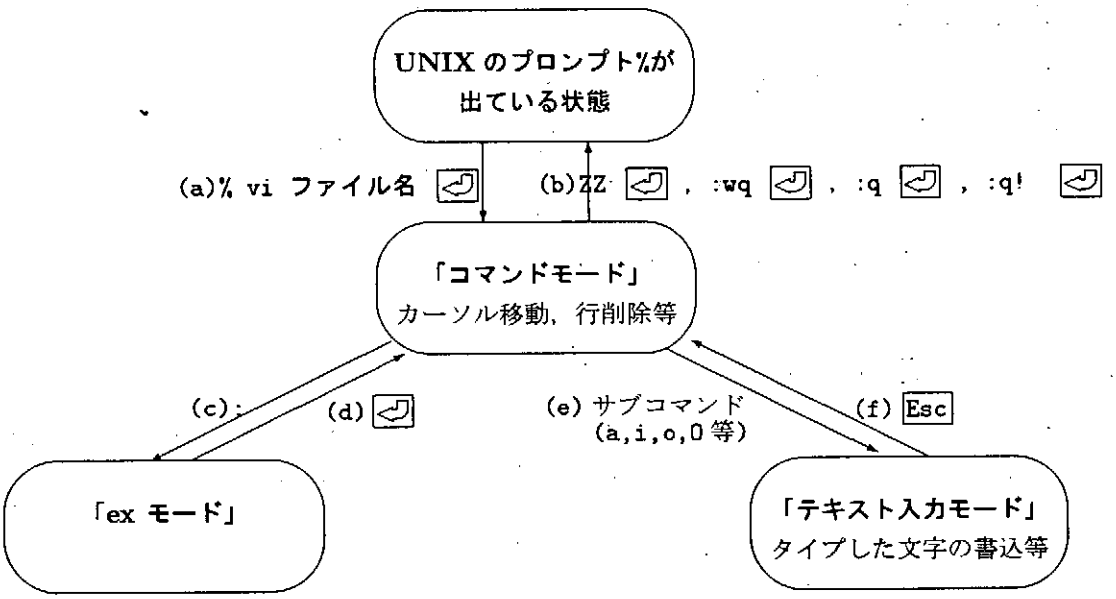


図 1.4: viのモード遷移.

(a) UNIXのプロンプト状態からコマンドモードへ

viの起動である.

```
$ vi ␣ ファイル名
```

(b) コマンドモードからUNIXのプロンプト状態へ

ファイルの保存やクイットである.

:ZZ , :wq , :q , :q! .

(c) コマンドモードからexモードへ

コマンドモードで‘‘ : "を入力する.

(d) exモードからコマンドモードへ

exモードでexコマンドを入力後, を入力する.

(e) コマンドモードからテキスト入力モードへ

文字の挿入(a,i,A,I)や行の挿入(o,O)である.

(f) テキスト入力モードからコマンドモードへ

文字列を入力後, Esc を押すとコマンドモードへ変わる. Esc が押されるまで, 入力した文字列が追加される.

1.8.3 基本的な使い方

それでは, ファイルopen.fに文字を入力する場合を例として, viの基本的な使い方について説明する.

(1) vi の起動 (プロンプト状態からコマンドモードへ)

ファイル名を指定して vi を起動する.

```
$ vi ␣ open.f ↵
```

(2) 書込み (コマンドモードからテキスト入力モードへ)

画面にプログラムを書き込む.

- **追加 (append)**

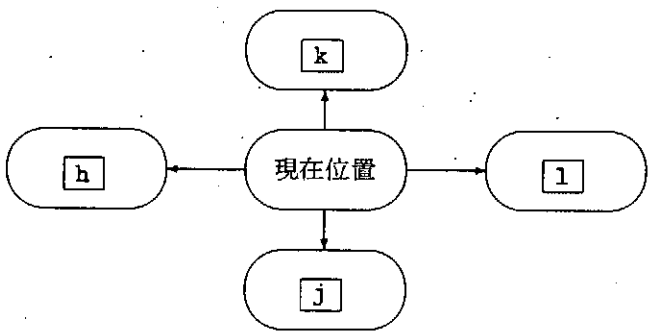
a 追加する文字列 Esc	Esc が押されるまで, 入力した文字列が追加される.

(3) 修正等 (コマンドモード)

修正等があればコマンドモードから行う.

- **カーソル移動**

カーソルの移動は次のように行う.



また, その他のカーソル移動は以下のように行う.

0	行頭 (第 0 カラム).
$	行末.
^	行頭からみて最初の空白以外の文字位置.
-	前の行の最初の空白以外の文字位置.
+	次の行の最初の空白以外の文字位置.
数字 G	**数字** 行目 (**数字** を省くと文末の行頭).

- **挿入 (insert)**

i **挿入する文字列** Esc	カーソルの前へ挿入.
a **挿入する文字列** Esc	カーソルの後へ挿入.
I **挿入する文字列** Esc	行頭へ挿入.
A **挿入する文字列** Esc	行末へ挿入.

- **行の挿入**

o **挿入する文字列** Esc	カーソルのある行の下へ挿入.
O **挿入する文字列** Esc	カーソルのある行の上へ挿入.

- **文字削除**

数字 x	カーソル位置から**数字**文字を削除.
数字 X	カーソル前の**数字**文字を削除.

どちらとも **数字** を省略すると **数字=1** と解釈される.

- **行の削除**

数字 dd	カーソル行から **数字** 行を削除.

数字 を省略すると **数字**=1 と解釈される.

- **取り消し**

u	直前の編集コマンドを取り消す.
U	現在行で行なわれたすべての編集作業を取り消す.

- **ex コマンド**

:*ex* **コマンド** ⏎	ex コマンド.

ex コマンド はあとで説明する.

(4) ファイルの保存やクイット (ex モード)

:ZZ ⏎	編集中のファイルをセーブしエディタから抜ける.
:wq ⏎	編集中のファイルをセーブしエディタから抜ける.
:w ␣ **ファイル名** ⏎	編集中のファイルを指定されたファイルに書き込む. **ファイル名** が省略された場合には, カレント・ファイルが指定されたものと解釈される.
:w ␣ >>**ファイル名** ⏎	編集中のエディタ・バッファの内容を指定されたファイルの後ろに追加する.
:q ⏎	エディタから抜ける.
:q! ⏎	編集内容を更新せずに, 強制的にエディタを終了する.

(5) ファイルの読込み (ex モード)

:r ␣ **ファイル名** ⏎	カーソル行の下に指定されたファイルを読み込む. **ファイル名** が省略された場合には, カレント・ファイルの内容を読み込む.
:r ␣ !*UNIX* **コマンド** ⏎	カーソル行の下に *UNIX* **コマンド** の実行結果 (標準出力) を読み込む.
!!*UNIX* **コマンド** ⏎	カーソル行以降に *UNIX* **コマンド** の実行結果 (標準出力) を読み込む. 最下行には, !*UNIX* **コマンド** と表示される.

1.8.4 発展的な使い方 (コマンドモード)

vi を用いてファイル編集を行う際に，更に快適にファイル編集を行うためのコマンドを説明する．

- **カーソル移動**

[数字] w	単語単位で右に移動．
[数字] W	スペースで区切られた単語単位で右に移動．
[数字] e	カーソルが置かれている単語の最後の文字に移動．
[数字] E	スペースで区切られた単語単位で最後の文字に移動．
[数字] b	単語単位で左に移動．
[数字] B	スペースで区切られた単語単位で左に移動．
%	対応する括弧に飛ぶ．ない場合，移動しない．
[数字] [	セクションの先頭へ移動．
[数字]]	次のセクションへ移動．
[数字] {	段落の先頭へ移動．
[数字] }	次の段落へ移動．
[数字] (	文章の先頭へ移動．
[数字])	次の文章へ移動．
[数字] H	Home Position (画面の上) から数えて下方に 数字 行目．
M	画面上の真ん中へ．
[数字] L	Last line (画面の下) から数えて上方に数字 行目．

[数字] を入力しないときは，1 と入力したのと同じ．

- **Joint**

J	カーソル行を前の行の後ろにつなげる．

- **探索 (Search)**

/文字列	文字列 の前方探索．
?文字列	文字列 の後方探索．
n	前回の search の繰り返し．
N	n の逆向き．

- **1 文字探索**

f 文字	カーソル位置から 文字を前方探索し，その位置へ．
F 文字	カーソル位置から 文字を後方探索し，その位置へ．
t 文字	カーソル位置から 文字を前方探索し，その一つ前の位置へ．
T 文字	カーソル位置から 文字を後方探索し，その一つ前の位置へ．

これらのコマンドの探索範囲はカーソルのある行のみ．

- **mark**

m 文字	現在位置にマーカー文字を付ける．ただし文字はアルファベットの小文字．
' 文字	マーカー 文字 のある行頭へ．
` 文字	マーカー 文字 の位置へ．

- **Delete**

[**"バッファ名**]**数字** dd	現在位置から **数字** 行削除する.
[**"バッファ名**]**数字** d [j,k etc.]	現在位置からカーソルモーション [j,k etc.] で指定した位置までを削除する.
[**"バッファ名**]dG	現在位置からファイルの終りまでを削除する.
[**"バッファ名**]d **数字** G	現在位置から **数字** 行目までを削除する.
[**"バッファ名**]d$	現在位置から行末までを削除する.
[**"バッファ名**] D	現在のカーソル位置から行末までを削除する.

初めの二つは, 繰り返し数 **数字** を省略した場合は **数字=1** として処理される. どの場合も削除した内容は **"バッファ名** で指定されるヤンク・バッファにコピーされる. ヤンク・バッファの名前が省略されるとテンポラリ・バッファが使われる. **バッファ名** はアルファベット小文字で1文字である.

- **Change**

[**"バッファ名**] **数字** cc	現在行から **数字** 行が, Esc が押されるまでに入力された文字列に置き換えられる.
[**"バッファ名**] **数字** c [j,k etc.]	現在位置からカーソルモーション [j,k etc.] を**数字** 回繰り返して到達する位置までが, Esc が押されるまでに入力された文字列に置き換えられる.
[**"バッファ名**] C	現在位置から行末までが, Esc が押されるまでに入力された文字列に置き換えられる.

初めの二つは, 繰り返し数 **数字** を省略した場合は**数字=1** として処理される.

- **Replace**

[**"バッファ名**] r **文字**	カーソル上の文字を **文字** に置き換える.
[**"バッファ名**] R **文字**	カーソル以降を Esc が押されるまでに入力された文字列で上書きする.

- **Substitute**

[**"バッファ名**]**数字** s **文字**	カーソル以降**数字**文字を Esc が押されるまでに入力された文字列に置き換える.
[**"バッファ名**] S	[**"バッファ名**] cc と同じ.

数字 を省くと **数字=1** として処理される.

置換コマンド r , R , s , S , cc , c ,C によって置き換えられた文字列 (元の文字列) は**バッファ名** で指定されたヤンク・バッファにコピーされる. ヤンク・バッファの名前が省略されるとテンポラリ・バッファが使われる.

- **Put**

[**"バッファ名**] p	**バッファ名** で指定されるヤンク・バッファの内容を現在のカーソル位置の後ろにコピーする. バッファの名前が省略されるとテンポラリ・バッファが使われる. カーソルの前にコピーするには大文字 P を使う.

- **Yank**

["バッファ名] **数字** yy	**バッファ名** で指定されるヤンク・バッファに，現在のカーソル行から **数字** 行をコピーする． yy の代わりに大文字 Y だけでも同じ動作をする．
["バッファ名] **数字** y [j,k etc.]	**バッファ名** で指定されるヤンク・バッファに，現在位置からカーソルモーション [j,k etc.] を **数字** 回繰り返して到達する位置までをコピーする．
["バッファ名] **数字** yw	**バッファ名** で指定されるヤンク・バッファに，現在位置から**数字** 個の単語をコピーする．

どちらの場合も **数字** を省くと **数字=1** として処理される．また，バッファの名前が省略されるとテンポラリ・バッファが使われる．

- **その他**

~	カーソル上の文字の大文字小文字の逆転．
>>	現在行のインデント 1 段深くする．
<<	現在行のインデント 1 段浅くする．
.	直前のコマンドを繰り返す．
[Ctrl] + *[l]*	画面が書き直される．
[Ctrl] + *[f]*	スクリーンに次のページを表示する．
[Ctrl] + *[d]*	前方スクロール．
[Ctrl] + *[b]*	スクリーンに前のページを表示する．
[Ctrl] + *[u]*	後方スクロール．
[q]	Open Mode に移る．

1.8.5 ex コマンド

ex モードは，ファイルへの保存やファイルの読込み，行指定や行に対する編集を行うことができる状態である．ex コマンドは，‘‘ : ’’ を押してから入力し，⏎ を押すことで利用できる．ここでは，よく利用する ex コマンドについて紹介する．

- **行の指定**

:**数字**	**数字** 行目．
:$	最後の行．
:.	現在の行 (指定しなくてもよい)．
:+	1 行先の行．
:**数字**+	**数字** 行先の行．
:-	1 行前の行．
:**数字**-	**数字** 行前の行．
:**数字** *1*, **数字** *2*	:**数字** *1* 行目から**数字** *2* 行目まで．
:%	初めから終わりまで．
:/**文字列**/	**文字列** のある行．
:' **文字**	マーカー **文字** を付けた行．

- **削除 (delete)**

:d	現在の行を削除する.
:**数字** *1*, **数字** *2* d	**数字** *1* 行目から **数字** *2* 行目まで削除する.
:**数字** *1* d **数字** *2*	**数字** *1* 行目から先 **数字** *2* 行を削除する.

- **表示 (print)**

:p	現在の行を表示する.
:**数字** *1*, **数字** *2* p	:**数字** *1* 行目から **数字** *2* 行目まで表示する.

- **置換 (substitute)**

:s/**文字列** *1*/**文字列** *2*/	最初に見つけた **文字列** *1* を **文字列** *2* に置き換える.
:**数字** *1*, **数字** *2* s/**文字列** *1* /**文字列** *2*/g	**数字** *1* 行目から **数字** *2* 行目までの中に**文字列** *1* を見つけたら **文字列** *2* に置き換える.
:%s/**文字列** *1*/**文字列** *2*/g	ファイル全体の **文字列** *1* を **文字列** *2* に置き換える.
:.,$s/**文字列** *1*/**文字列** *2*/g	現在行からファイルの最後までの**文字列** *1* を **文字列** *2* に置き換える.

- **移動 (move)**

:**数字** *1*, **数字** *2* m **数字** *3*	**数字** *1* 行目から **数字** *2* 行目までを **数字** *3* 行目の後に移動する.

- **コピー**

:**数字** *1*, **数字** *2* t **数字** *3*	**数字** *1* 行目から **数字** *2* 行目までを **数字** *3* 行目の後に複写する.

- **エディタ外コマンドの実行**

:! *UNIX* **コマンド**	UNIX のコマンドをバック・グラウンドで実行する.

- **マクロ**

:map ␣ **記号** ␣ **機能**	**記号** をマクロ名として, **機能** を登録する. **記号** としては, #a のような, **#文字** を勧める.
:unmap ␣ **記号**	マクロ名 **記号** を消去する.
:map	マクロ名と機能を表示する.

- **オプションの設定**

:set	設定されているオプションが表示される.
:set ␣ number	行番号が表示される.
:set ␣ nonumber	行番号が削除される.
:set ␣ **オプション名**	オプションが設定される.
:set ␣ no **オプション名**	オプションが設定が解除される.
:set ␣ all	使用できるオプションが表示される.

- その他

:e ␣ ファイル名	指定されたファイルの編集に移る.
:e! ␣ ファイル名	現在編集中の内容を放棄して指定されたファイルのエディットに取り掛かる.
:args	コマンドライン上で与えられた編集の一覧を表示する.
:e#	コマンドライン上で与えられた前のファイルの編集に移る.
:n	コマンドライン上で与えられた次のファイルの編集に移る.
:n!	現在編集中の内容を放棄して, コマンドライン上で与えられた次のファイルの編集に移る.
:rew	最初のファイルに戻る.
:rew!	現在編集中の内容を放棄して, 最初のファイルに戻る.

1.9 データのバックアップ

1.9.1 MS-DOS フロッピ・ディスク (mtools)

UNIX では, MS-DOS フロッピ・ディスク (2HD 1.44MB , 2DD 720kB) を読み書きするためのコマンド mtools が用意されている.

- ファイルの読み込み

  フロッピ・ディスク上に保存されているファイルを読み込むには, プロンプトに対し,

  $ mread ␣ -t ␣ *MS-DOS* ディスクでのファイル名 ␣ [*UNIX* でのファイル名] ⏎

  と入力する. バイナリの時にはオプション -t はいらない. *UNIX* でのファイル名 を省略した場合には, *MS-DOS* ディスクでのファイル名 と同じになる.

- ファイルの書き込み

  フロッピ・ディスクにファイルを書き込むには, プロンプトに対し,

  $ mwrite ␣ -t ␣ *UNIX* でのファイル名 ␣ [*MS-DOS* ディスクでのファイル名] ⏎

  と入力する. バイナリの時には -t はいらない. *MS-DOS* ディスクでのファイル名 を省略した場合には, *UNIX* でのファイル名 と同じになる.

- フロッピの内容の表示

  フロッピ・ディスクの内容を表示するためには, プロンプトに対し,

  $ mdir ⏎

  と入力する.

- ファイルの消去

  フロッピ・ディスク上のファイルを消去するには, プロンプトに対し,

  $ mdel ␣ ファイル名 ⏎

  と入力する.

- MS-DOSフロッピ・ディスクの取り出し

  フロッピ・ディスクをフォッピ・ディスク・ドライブから取り出すには，プロンプトに対し，

```
$ eject ↵
```

  と入力する．

- mreadでのワイルド・カードの使い方

  例えば，open*という複数のファイルを読み込むときには，プロンプトに対し

```
$ mread ␣ -t ␣ exam"*" ␣ . ↵
```

  と入力する．これは，ワイルド・カードはふつうカレント・ディレクトリを読んで補完するためである．

1.9.2 mtoolsコマンドのまとめ

mread ␣ *MS-DOSファイル名* ␣ **ファイル名** *(または .)*	読み込み
mwrite ␣ **ファイル名**	書き込み
mdel ␣ **ファイル名** ␣ *MS-DOSファイル名* *(または .)*	ファイルの消去
mdir	ファイル名の表示
mcd ␣ **ディレクトリ名**	ディレクトリの移動

1.9.3 バックアップの手順

ここで，1.6.8で紹介した圧縮のためのコマンドgzipと1.6.8で紹介した複数のファイルを一つにまとめるコマンドtarを用いて，ファイルをバックアップする手順を説明する．

- **バックアップする**

  たとえば，~open00/test以下にあるファイルをバックアップする場合を考える．

```
$ cd ␣ ~open00 ↵
$ ls ↵
test
$ ls ␣ test ↵
file1.f   file2.f
```

  まず，ディレクトリtestにtarコマンドを用いる．

```
$ tar ␣ cvf ␣ back.tar ␣ ./test ↵
./test/
./test/file1.f
./test/file2.f
$ ls ↵
back.tar   test
```

  生成されたback.tarをgzipで圧縮する．

```
$ gzip ␣ back.tar ↵
$ ls ↵
back.tar.gz  test
```

MS-DOS ファイルは，8 文字と . と拡張子 3 文字で名付けられるので，プロンプトに対し

```
$ mv ␣ back.tar.gz ␣ back.tgz ↵
$ ls ↵
back.tgz  test
```

と名前を変更する．ここで，.tgz という拡張子は，tar して gzip したファイルと認識される．

あとは，この back.tgz をフロッピ・ディスクに書き込む．

```
$ mwrite ␣ back.tgz ␣ . ↵
$ mdir ↵
back.tgz
```

- **バックアップを戻す**

たとえば，~open00 以下にフロッピ・ディスク上の back.tgz を戻す場合を考える．

フロッピ・ディスクからファイルを読み込む．

```
$ cd ␣ ~open00 ↵
$ mdir ↵
back.tgz
$ mread ␣ back.tgz ␣ . ↵
$ ls ↵
back.tgz
```

back.tgz を gzip で解凍する．

```
$ gzip ␣ -d ␣ back.tgz ↵
$ ls ↵
back.tar
```

解凍された back.tar を展開する．

```
$ tar ␣ -xvf ␣ back.tar ↵
./test/file1.f
./test/file2.f
$ ls -F ↵
back.tar  test/
$ ls ␣ test ↵
file1.f  file2.f
```

1.10 Fortranでのコンパイルと実行の制御

1.10.1 コンパイル

エディタを使って書かれたソースコードはそのままでは実行することはもちろんできない．ソースコードを計算機が実行できる形式に変換する作業が**コンパイル**である．

```
$ efc ␣ ソースコード ⏎
```

(この教育講座で用意したFortranコンパイラーはefcである.) この作業によって，**ソースコード**.*o*という**オブジェクトファイル**と，a.outという**ロードモジュール**ができる．

1.10.2 コマンドの実行

f90(efc)，cc 等によって作られた a.out のようなロード・モジュールの実行にはロード・モジュール名をコマンド名と同様に扱うことができる(**フォア・グラウンド**)．たとえば，プロンプトに対し，

```
$ ./a.out ⏎
```

とタイプすればよい．

1.10.3 フォア・グラウンドとバック・グラウンド

コマンドやジョブの実行は，フォア・グラウンドで行う場合と**バック・グラウンド**で行う場合がある．ロード・モジュール a.out をバック・グラウンドで実行するには，プロンプトに対し

```
$ a.out ␣ & ⏎
```

とタイプすればよい．

1.10.4 ジョブの確認

現在のジョブの状況を見るには，プロンプトに対し

```
$ps ␣ [オプション] ⏎
```

とタイプすればよい．これにより，login した後の実行中，停止中の自分のジョブの状況を見ることができる．その際，出てくる番号が**ジョブ番号**である．この番号はそのユーザーのジョブに対して割り当てられた番号である．また，オプションを指定することで，loginする前からバック・グランドで流れているジョブを含む現在実行中・停止中の自分のジョブの状況や，システム全体で流れているジョブの状況を表示することができ，その際，システム全体を通じてつけられた番号(**プロセス番号**)も表示することができる(詳細はマニュアル参照)．

1.10.5 ジョブの強制終了

- **フォア・グラウンドで現在実行中のジョブの場合**

```
Ctrl + c
```

を押す．

- **現在停止中またはバック・グラウンドで実行中の場合**
  プロンプトに対し

```
$ kill ␣ %ジョブ番号 ⏎
```

  または,

```
$ kill ␣ プロセス番号 ⏎
```

  とタイプする.

1.10.6 ジョブの一時停止

- **フォア・グラウンドで現在実行中のジョブの場合**

```
Ctrl + z
```

  を押す.

- **カレント・セッション (バックグラウンドで走らせてから logout していない) 場合**
  プロンプトに対し

```
$ kill ␣ -STOP ␣ [% ジョブ番号] ⏎
```

  または,

```
$ stop ␣ [%ジョブ番号] ⏎
```

  とタイプする. **%ジョブ番号** を省略したときには, 直前にバックグラウンドにしたものが指定される.

- **カレント・セッションでない (バックグラウンドで走らせ logout 後に, 再び login した) 場合**
  プロンプトに対し

```
$ kill ␣ -STOP ␣ プロセス番号 ⏎
```

  または

```
$ stop ␣ プロセス番号 ⏎
```

  とタイプする.

1.10.7 カレント・セッションで一時停止したジョブの再開

- **フォア・グラウンド・ジョブとして再開する場合**
  プロンプトに対し

```
$ fg ␣ [%ジョブ番号] ⏎
```

  または

```
$ジョブ番号 ↵
```

とタイプする.

- **バック・グラウンド・ジョブとして再開する場合**
  プロンプトに対し

```
$ bg ␣ [%ジョブ番号] ↵
```

または

```
$ジョブ番号 ␣ & ↵
```

とタイプする.
どちらの場合も, **%ジョブ番号** を省略すると, 一番新しく停止させたジョブが指定される.

1.10.8 一時停止したプロセスの再開

プロンプトに対し

```
$ kill ␣ -CONT ␣ プロセス番号 ↵
```

とタイプする.

1.11 使ってみよう

UNIX の使い方 (例)
起動 から 終了 まで

1. **Windows** : Windows の起動.
2. **Cygwin** : Cygwin の起動. Cygwin アイコンをダブルクイック.
3. **startx** : X Window System の起動. Cygwin ターミナルで

```
$ startx
```

4. **ssh** : リモートシステムへのログイン.

```
$ ssh open00@remote.machine.name
open00@remote.machine.name's password:********
%
```

5. **pwd** : カレントディレクトリ名を出力 (print working directory)

```
% pwd
/localuser/open00
```

6. **mkdir** : ディレクトリの作成 (make directory)

```
% mkdir example
```

7. **ls** : ファイルやディレクトリの情報出力 (list)

```
% ls
example
% ls -F
example/
```

8. **cd** : ディレクトリの変更 (change directory)

```
% cd example
% pwd
/localuser/open00/example
% cd ..
```

9. **rmdir** : ディレクトリの消去 (remove directory)

```
% rmdir example
% ls
```

10. **mkdir, cd, vi** : ディレクトリの作成及び変更, ファイルの作成 (あるいは, mule など)

```
% mkdir text
% ls text
% cd text
% pwd
/localuser/open00/text
% vi test.txt

i
This is a text file.
Esc
:ZZ
```

11. **cat** : ファイル内容の出力(concatenate)(あるいは, more など)

```
% cat test.txt
This file is a text file.
```

12. **mv** : ファイルの移動(move)

```
% mv test.txt test1.txt
% ls
test1.txt
% cat test1.txt
This file is a text file.
```

13. **cp** : ファイルの複写(copy)

```
% cp test1.txt test2.txt
% ls
test1.txt  test2.txt
% cat test2.txt
This file is a text file.
```

14. **rm** : ファイルの消去(remove)

```
% rm -i test2.txt
rm: remove test2.txt? y
% ls
test1.txt
```

15. **cd** : ホームディレクトリへの移動(change directory)

```
% cd
% pwd
/localuser/open00
```

16. **tar, gzip, rm** : ファイルの圧縮及び不要なファイルの消去, 展開

```
% ls
text
% tar cvf text.tar ./text
./text/
./text/test1.txt
% ls
text text.tar
% ls -l
-rw------- open00 openlab    10240 Oct 14 13:30 text.tar

text:
drwx------ open00 openlab        0 Oct 14 13:30 ./
drwx------ open00 openlab        0 Oct 14 13:30 ../
-rw------- open00 openlab       20 Oct 14 13:30 test1.txt
% gzip text.tar
% ls -l
-rw------- open00 openlab      193 Oct 14 13:30 text.tar.gz

text:
drwx------ open00 openlab        0 Oct 14 13:30 ./
drwx------ open00 openlab        0 Oct 14 13:30 ../
-rw------- open00 openlab       20 Oct 14 13:30 test1.txt
% rm -r text
rm: descend into directory 'text'?  y
rm: rmove 'text/test1.txt'?  y
rm: rmove directory 'text'? y
% ls
text.tar.gz
% tar xvzf text.tar.gz
./text/
./text/test1.txt
% ls
text text.tar.gz
% ls text
test1.txt
```

ここで行なった tar と gzip での圧縮は

```
% tar cvzf text.tgz ./text
```

というように一度に行なうことができる.

17. 教育講座の準備：必要なファイルをコピー及び展開

```
% cp ~open00/virpex1.tgz ~/
% ls
text  text.tar.gz  virpex1.tgz
```

```
% tar xvzf virpex1.tgz
......
% ls
text  text.tar.gz  virpex1  virpex1.tgz
```

~open00/mhd2003.tgz も同様にコピーして展開する.

18. Fortran の利用例

(a) **cd** : 作業ディレクトリへの作成と変更

```
% mkdir sample
% cd sample
% pwd
/localuser/open00/sample
% ls
```

(b) **vi etc** : ファイルの書込み

```
% vi sample.f
% ls
sample.f
% cat sample.f
      implicit none
      real(kind=8):: x,y,z
      read(5,*) x,y
      z=x+y
      write(6,*) z
      stop
      end
```

(c) **efc** : Fortran プログラムのコンパイル (あるいは, f77, f90 など)

```
% efc sample.f
% ls -F
a.out*  sample.f sample.o
```

(d) **a.out etc** : 実行型ファイルの実行

```
% ./a.out
4.5
1.9
    6.400000000000000
```

(e) ファイルの整理

rm : 不要なファイルを消去する

```
% rm a.out sample.o
rm: remove a.out? y
rm: remove sample.o? y
```

gzip etc ： しばらく使用しないファイルを圧縮する

```
% gzip sample.f
% ls
sample.f.gz
```

gzip etc ： 圧縮したファイルを解凍する

```
% gzip -d sample.f.gz
% ls
sample.f
```

(f) **cd** ： ホームディレクトリに変更する

```
% cd
% pwd
/localuser/open00
% ls
text  text.tar.gz  sample  virpex1  virpex1.tar.gz
```

19. **logout** ： リモートシステムからのログアウト.

```
% logout
```

20. **sftp** ： 計算機サーバからファイルを転送.

```
$ sftp open00@remote.machine.name
open00@remote.machine.name's password:********
sftp> ls
text
text.tar.gz
sample
virpex1
virpex1.tar.gz
sftp> get text.tar.gz
text.tar.gz                    100% 193    0.9KB/s   00:00
sftp> exit
$ ls
text.tar.gz
$ tar xvzf text.tar.gz
./text/
./text/test1.txt
$ ls
text text.tar.gz
```

21. **exit** ： X Window System の終了：ログインターミナル上で.

```
$ exit
```

22. **exit** : Cygwin の終了.

```
$ exit
```

23. **Windows** : Windows の終了.

関連図書

[1] 斎藤信男編:岩波コンピュータサイエンス「ユーザーズ UNIX」(岩波書店)1988.

[2] 山口和紀, 古瀬一隆監修:「新 The UNIX Super Text(改訂増補版)」(技術評論社)2003.

[3] 林晴比古:「新 Linux/UNIX 入門」(ソフトバンクパブリッシング)2000.

[4] 坂本文:「たのしい UNIX--UNIX への招待--」(アスキー)1990.

[5] 坂本文:「続・たのしい UNIX--UNIX への招待--」(アスキー)1993.

[6] 慶應義塾大学理工学部物理学科理論研究室編 : UNIX 初期教育資料.

[7] 慶應義塾大学理工学部計算センター発行 :「UNIX ワークステーションを使ってみよう!」1993.

[8] 慶應義塾大学理工学部計算センター発行 :「UNIX ワークステーションをもっと使ってみよう!」1994.

[9] 佐藤竜一, いけだやすし, 野村直:「Cygwin+JE--Windows で動かす UNIX 環境」(アスキー)2003.

[10] 中村繁利, 熊谷直樹, 御影伸哉:「Windows 上で実現される UNIX 環境--Cygwin を使おう」(ディー・アート)2003.

[11] 川井義治, 米田聡:「Cygwin--Windows で使える UNIX 環境」(ソフトバンクパブリッシング)2002.

[12] D. ゾンネンシャイン著, 武田英明訳:「UNIX スクリーンエディタ vi ガイド」(啓学出版)1988.

[13] 矢吹道郎監修:「初めて使う GNU Emacs(啓学出版)1992.

[14] R.M.Stalllman 著, 竹内郁雄, 天海良治訳:「GNU Emacs マニュアル」(共立出版)1988.

第2章　FORTRAN概説

2.1　言語仕様

パーソナルコンピュータ（パソコン）が身近に存在し、手軽に利用することができるようになった現在、皆さんはコンピュータ（ハードウェア）がどのようにして動作しているのかあまり意識していないかもしれませんが、アプリケーションとかプログラムとかコードとか呼ばれるソフトウェアによってハードウェアが制御されている[1] ということを知っている方は多くいらっしゃると思います。

それでは、そのソフトウェアはどのような形でコンピュータの中に格納されているのでしょうか。日本語でしょうか、それとも人間にとって理解しやすいそれ以外の言語でしょうか。実は、コンピュータが理解できるコード（機械語）によって格納されているのです。それは、コンピュータが直接理解することができるのは機械語だけだからなのです。

機械語は通常二進数（0 と 1 の値だけをとる）の数値がある規則に従って並べられた形になっています。コンピュータにとっては分かりやすい（直接実行できる）形式ですが、人間から見るとただの数値の羅列のように見え、そのプログラムがどのように動作するのかを理解することは非常に困難です。もし、そのプログラムの中に間違いがあったとしても、それを発見し修正することは容易ではありません。そこで、二進数の機械語をもう少し可読性を持たせた形で表現したニーモニックという表記法もあります。しかし、これも所詮機械語を少々見やすくしただけのものであって、万人が容易に理解できるようなものではありません。

コンピュータの黎明期には、このような機械語またはニーモニックを用いてプログラムを書くことが必要とされていました。したがって、だれもが簡単に利用できるようなものではなく、ちょっとした計算などを行うだけでもかなりの努力を必要とするような状況にあり、コンピュータを利用することには大きな障壁が存在したのです。

この状況を打破するべく登場したのが FORTRAN でした。1956 年に IBM 社から発表されたこの FORTRAN (FORmula TRANslation:　数式翻訳）という言語によって、それまでに存在したプログラミングの障壁は低いものとなり、比較的容易にコンピュータを利用することが可能となったのです。この FORTRAN がそれまでと異なっている点は、算術計算などの処理中に含まれる手続きを日常（人間が）用いる表現[2] に近い形で整理した言語になっている点にあります。FORTRAN で書かれているプログラム自身をコンピュータが直接解釈・実行することはできませんから、コンパイラ（compiler）と呼ばれるプログラムによって FORTRAN で書かれたプログラムを機械語に翻訳し、その生成された機械語コードを実行するという手順をとることになります。

この FORTRAN の発表により、IBM のコンピュータは売り上げを大きく伸ばしました。当然、他のコンピュータ会社も IBM をまねて FORTRAN コンパイラを開発していきます。しかしながら標準規約といったものが存在しなかったため、方言ともいえるような似て非なる FORTRAN 言語がたくさん存在する状況になっていってしまいました。

この状況を打破するために標準化作業が行われ、これまでに大きく分けると「FORTRAN66 言語仕様（規

[1] 『コンピュータ、ソフトなければただの箱』などと言われることもありますね
[2] ただし、残念ながら日本語ではなく英語がベースとなっています

格）[3] 」、「FORTRAN77 言語仕様（規格）[4] 」、「Fortran90(95) 言語仕様（規格）[5] 」が制定されてきました[6] 。さらに、「Fortran2000 規格」を制定する方向で活動が行われています。

「FORTRAN66 言語仕様」は、それまでに存在していた FORTRAN の方言を統一するための非常に古い言語仕様であり、今では用いられることはあまりありません。「FORTRAN77 言語仕様」は、ベンダ各社が汎用的にサポートしてきていた仕様およびいくつかの新しい機能を取り入れた仕様で、長い間科学技術計算コードの分野で広く用いられてきています。古い言語仕様ではありますが、いまだに現役として使用されています。「Fortran90(95) 言語仕様」は、「FORTRAN77 言語仕様」を拡張した仕様になっており、配列の動的確保、ポインター、配列演算の記述法などが特徴的なものとなっています。優れたコンパイラが発表されてきている[7] こともあって、徐々に「FORTRAN77」からの移行が進んできています。

本テキストは大部分が「FORTRAN77 言語仕様」をベースにして記述してありますが、「Fortran90 言語仕様」の特徴も取り入れた構成になっており、一部の FORTRAN77 コンパイラ（処理系）には対応しない記述がある場合がありますのでご注意ください。

2.2 FORTRAN プログラムの形式

FORTRAN プログラムは、文（statement）と行（line）とから構成されています。FORTRAN プログラムには、大きく分けると 2 つの形式があり、それぞれ固定形式（fixed source form）および自由形式（free source form）と呼ばれています。自由形式は Fortran90 で正式に認められるようになった新しい形式であり、従来からの FORTRAN プログラマーは固定形式でプログラムを記述することが多いようです。ひとつのプログラムの中で混在させることは許されていないので、どちらを使うか決めないとなりませんが、それぞれ特徴があり、どちらが優れていると判断することはできないので、好みで選べばよいと思いますが、これから新たに Fortran を学ぼうという方は、自由形式を選択しておいたほうがベターであると思います。それぞれの形式について簡単にまとめると次のようになります。

固定形式

- 1 行の長さは 72 文字で、行の長さが 72 文字に満たない場合は残り部分に空白があるとみなされます [8]
- 各文（statement）は各行（line）の桁（column）7〜72 の間に書かなければなりません [9]
- 1 つの文が数行にわたる場合、継続行 [10] の桁 6 に '0' および空白以外の文字を書き、桁 1〜5 は空白にしておかなければなりません
- 一つの文は最大 20 行まで続けて書くことができます

[3] ANSI X3.9-1966, JIS X 3001-1976
[4] ISO 1539:1980(E), ANSI X3.9-1978,JIS X 3001-1982
[5] ISO/IEC 1539 : 1991(E), ANSI X3.198 - 1992, JIS X 3001-1994, ISO/IEC 1539-1:1997, JIS X 3001-1:1998
[6] "FORTRAN"と"Fortran"が使われていることに気づくと思いますが、これは Fortran90 になって正式に小文字の使用が許されるようになったという事実を反映しているようです
[7] ただし残念なことに、無料で使える Fortran90 コンパイラの種類は非常に限定されています
[8] 1 行の長さが 72 文字に制限されているのは、プログラムの入力にカードを使っていたことの名残りであると思われます
[9] 固定形式では 1〜6 カラムには特別の意味があります
[10] 第一行目を開始行、それに続く行を継続行とよびます

自由形式

- 1行の長さは最大132個の文字を含むことができます
- 先行空白には意味がないので、全ての文を行の先頭位置から書き始めることも、利用者の選んだ配置に合わせて任意の文字位置から書き始めることもできます
- 追加の継続行は最大39行まで許され、継続記号 '&' を付加して継続行であることを指示します
- 注釈行ではない行において、& が空白でない最後の文字である場合、その文は & の直前の文字から次の行に続き、特に次の行の空白でない最初の文字が & である場合には、& の次の文字に続きます

Tips: 固定形式と自由形式との共存

基本的には固定形式と自由形式とを共存させることはできませんが、裏技的な方法として

- プログラム自体は固定形式同様第7カラムから第72カラムまでの間に記述する
- 行番号は第1から第5カラムに配置する
- 注釈は '!' を用いて記述する
- 文の継続は継続される行の第73カラムおよび継続する行の第6カラムに '&' を記述する

ことによって、固定形式としても自由形式としても(処理系によっては)扱うことが可能となります。

2.3 注釈行

注釈行というのは、人間がプログラムを見たときに理解しやすくするために挿入するものであり、コンパイラ(コンピュータ)にとっては全く意味を持たず、翻訳時には無視されます。しかし、だからといって無駄なものではなく、後でプログラムを見たときにその動作が容易に理解できるよう、積極的に適切に注釈を加えておくことはプログラミングにとって非常に重要なことです[11]。

固定形式の場合

桁1に 'C' または '*' を書くことによって、その行が注釈であることを表します。また、桁1から72まで全て空白の行も注釈行と解釈されます。桁1が 'C' または '*' である注釈行の桁2〜72には、処理系で扱える全ての文字を使うことができます。注釈行は、プログラム中の任意の場所に書くことができ[12]、翻訳時には無視されます。

自由形式の場合

'!' 以降行末までは注釈であるとみなされ、翻訳時には無視されます。また、空行も注釈行と解釈されます。

処理系によっては、固定形式であっても '!' を注釈文字であると解釈します。

[11] 適切な注釈を記述することはプログラミングの一部であると考える人もいます
[12] 複数の継続行の中に注釈行を入れることも認められている

2.4 文番号

文には、それを他の文から参照できるようにするため番号（文番号）を付けることができます。文番号には1から99999の自然数を用い、固定形式の場合にはその文の開始行の桁1〜5に書きます。自由形式の場合には、文の先頭に書きます。文番号の大小は文の順番に従う必要はありませんが、一つのプログラム単位中の2つ以上の文に同じ文番号を付けてはいけません。文番号の頭に'0'を付けることや空白を入れることができますが、それらは無視されます[13] 。文番号を参照するケースとしては、Doループのラベル、Format文のラベル、Go to文の飛び先などがありますが、プログラム構造を工夫すれば文番号を用いなくてもプログラミングが可能です。

例外として、入力文のエラー処理の飛び先指定をするには文番号を用いる必要があります。

2.5 文の種類とその順序

<table>
<tr><td colspan="3">Program 文、 Function 文、 Subroutine 文、 Module 文</td></tr>
<tr><td colspan="3">Use 文</td></tr>
<tr><td rowspan="4">Format 文</td><td colspan="2">Implicit none 文</td></tr>
<tr><td>Parameter 文</td><td>Implicit 文</td></tr>
<tr><td>Parameter 文
Data 文</td><td>構造型定義
引用仕様宣言
型宣言文
単純宣言文</td></tr>
<tr><td colspan="2">実行文（代入文、制御文、入出力文）</td></tr>
<tr><td colspan="3">Contains 文</td></tr>
<tr><td colspan="3">内部プログラム
モジュール副プログラム</td></tr>
<tr><td colspan="3">End 文</td></tr>
</table>

2.6 実行可能プログラムの単位

プログラム単位 (program unit)（文と注釈行とから構成される）

- 主プログラム （Program 文で始まるプログラム単位）
  - 内部副プログラム（Contains 文で始まる）
    - 関数副プログラム（FUNCTION 文で始まる）
    - サブルーチン副プログラム（SUBROUTINE 文で始まる）

外部副プログラム

- 手続き副プログラム
  - 関数副プログラム（FUNCTION 文で始まるプログラム単位）
  - サブルーチン副プログラム（SUBROUTINE 文で始まるプログラム単位）
- 初期値設定副プログラム（BLOCK DATA 文で始まるプログラム単位）

[13] ␣␣10，␣␣10␣，␣␣1␣0，00010 は全て、同じ文番号 '10' をもつと見なされます

2.7 FORTRAN で使える文字

FORTRAN で使える文字は、英字、数字、特殊文字である。

英字 (letter)	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
	a b c d e f g h i j k l m n l p q r s t u v w x y z
数字 (digit)	0 1 2 3 4 5 6 7 8 9
特殊文字	␣ = + - * / ( ) , . ' " :; < > % ? ! _ ¥ $

これら以外の文字も処理系によっては使用可能である。
英字、数字の大小順序は、␣< A < B < C < ··· < Y < Z , ␣ < 0 < 1 < ··· < 8 < 9 である[14] 。

2.8 データの型

FORTRAN では、次の 6 種のデータ型を扱うことができる。

1. **整数型** −2147483648 から 2147483647 までの整数
2. **実数型** 有効数字 7 桁以内の符号付き実数および実指数部 (E の後ろに 2 桁以内の整数を続けたもの)。絶対値の範囲は 0 あるいは 10^{-78}〜10^{75} の範囲になければならない。
3. **倍精度実数型** 有効数字 16 桁以内の符号付き実数および倍精度指数部 (D の後ろに 2 桁以内の整数を続けたもの)。絶対値の範囲は 0 あるいは 10^{-78}〜10^{75} の範囲になければならない。
4. **複素数型** 実部および虚部をコンマで区切り全体を括弧でくくったもの。実部および虚部は実数型か整数型。
5. **論理型** .TRUE. および .FALSE. の 2 つ。
6. **文字型** 処理系で使用できる文字 (character)。

それぞれのデータ型は異なるものであり、内部表現も異なる。型は、そのデータを含む演算の解釈に影響を及ぼすため注意する必要がある。
型を指定するためには次の 3 つの方法がある;

1. **型宣言文を用いる方法**
2. **IMPLICIT 文を用いる方法**
3. **暗黙の型宣言を用いる方法** 名前の先頭文字が I, J, K, L, M ,N なら整数型、それ以外は実数型

が、きちんと型を把握しておくことが望ましく、またタイプミスがあってもコンパイル時に発見できるということから、

```
␣␣␣␣␣␣IMPLICIT␣NONE
*␣␣␣␣␣全ての定数、変数、配列に対する型宣言文
```

とすることを推奨する[15] 。

[14] 英字と数字間の大小順序は文字コード系によって異なる。また、文字コードの連続性は保障されていない。
[15] この書き方は、FORTRAN77 ではなくて fortran90 で規定された書き方である

2.9 配列

配列 (array) は、1次元から7次元までの順序づけられた値の集合であり、配列名によって識別する。配列を使用する場合には、そのプログラムのはじめの部分で型宣言文、COMMON文、DIMENSION文のいずれかにより配列宣言子を用いて、配列名、次元数、各次元の寸法を規定する。

配列宣言子 $a(d[,d][,d][,d][,d][,d][,d])$
a は配列名、d は寸法宣言子。
寸法宣言子 $[d_1:]d_2$
d_1 は寸法の下限、d_2 は寸法の上限で、いずれも算術式かつ $d_1 \square d_2$。ただし、式中の定数、定数名、変数は全て整数型とし、関数や配列要素の引用は不可。下限が省略された場合は、$d_1 = 1$ とみなす。

寸法の大きさは、$d_2 - d_1 + 1$ で、配列の大きさ (配列の要素の個数) は配列宣言子で指定した各次元の寸法の大きさの積になる。

配列要素 (array element) は、配列を構成している要素のおのおのである。配列要素は配列要素名によって識別する。

配列要素名 $a(s[,s][,s][,s][,s][,s][,s])$
a は配列名、$(s[,s][,s][,s][,s][,s][,s])$ は**添字 (subscript)**、s は**添字式**。添字式は整数式で、配列要素名や関数の引用を含んでもよい。各添字式の値は、その配列に対する配列宣言子の対応する下限以上、上限以下でなければならない。

添字の値 (subscript value) によって配列要素名の識別する配列要素が決定される。添字の値は下表のようにして定められる。

次元数	寸法宣言子	添字	添字の値
1	$(j_1 : k_1)$	(s_1)	$1 + (s_1 - j_1)$
2	$(j_1 : k_1, j_2 : k_2)$	(s_1, s_2)	$1 + (s_1 - j_1) + (s_2 - j_2) * d_1$
3	$(j_1 : k_1, j_2 : k_2, j_3 : k_3)$	(s_1, s_2, s_3)	$1 + (s_1 - j_1) + (s_2 - j_2) * d_1 + (s_3 - j_3) * d_2 * d_1$
⋮	⋮	⋮	⋮

ここで、$d_i \equiv k_i - j_i + 1$ であり、i 番目の寸法の大きさを表す。

例: 配列宣言子 `C(2,3)` によって宣言された配列 `C` の配列要素の順序は `C(1,1)`, `C(2,1)`, `C(2,1)`, `C(2,2)`, `C(1,3)`, `C(2,3)` になり、添字の値はそれぞれ `1`, `2`, `3`, `4`, `5`, `6` となる。

例: 配列宣言子 `C(-3:3)` によって宣言された配列 `C` の配列要素 `C(1)` の添字は (1)、添字式の値は1であり、添字の値は5である。ここで、添字の値と添字式の値とが必ずしも一致しないことに注意。

部分列: 順序づけられた集合を**列 (sequence)** という。列は空集合でもよい。文字データは空でない文字の列であり、文字データの連続した部分を部分列 **(substring)** または文字部分列といい、部分列名で識別する。

部分列名 $v([e_1]:[e_2])$ または $a(s[,s]...)([e_1]:[e_2])$
v は文字変数名、$a(s[,s]...)$ は文字配列要素名。e_1 および e_2 は文字位置式とよばれる整数式で、値 e_1 により部分列の左端の文字位置を、値 e_2 により部分列の右端の文字位置を指定する。文字変数または文字配列要素の長さを len とするとき $1 \square e_1 \square e_2 \square len$ でなければならない。文字部分列の長さは、$e_2 - e_1 + 1$ である。e_1 省略時は $e_1 = 1$ とみなし、e_2 省略時は $e_2 = len$ とみなす。両方の省略も可。$v(:)$ は v と、$a(s[,s]...)(:)$ は $a(s[,s]...)$ と同じ。

2.10 式

2.10.1 算術式

算術式は、算術演算を表現するために用い、評価すると数値が得られる。
算術演算子とその優先順位は次のとおり。ただし、演算順序が括弧によって示されている場合はそれに従う。

演算子	優先順位
**(冪乗)	高
/(除算), *(乗算)	｜
-(減算、符号反転), +(加算)	低

算術式の構成規則

1. 算術定数、算術定数名、算術変数、算術配列要素、算術関数の引用は算術式
2. 算術式を括弧で括ったものも算術式
3. 算術式を算術演算子で結合させたものも算術式[16]
4. 算術式の先頭に符号 (+または −) をつけることができる
5. 冪乗演算が２回以上続く場合、右から左へ結合される[17]
6. 乗算または除算が２回以上続く場合、左から右へ結合される[18]
7. 加算または減算が２回以上続く場合、左から右へ結合される[19]

算術式の型と解釈

- 単独の被演算子に+または − の演算子を施した式の結果の型は、その被演算子の型と同じ
- 加算、減算、乗算、除算の結果の型および解釈は下表の通り。⊙は演算子+,-,*,/を表し、I, R, D, C はそれぞれ整数型、実数型、倍精度実数型、複素数型を表す。

X_1	X_2			
	I_2	R_2	D_2	C_2
I_1	$I=I_1 \odot I_2$	$R=REAL(I_1) \odot R_2$	$D=DBLE(I_1) \odot D_2$	$C=CMPLX(REAL(I_1, 0.) \odot C_2$
R_1	$R=R_1 \odot REAL(I_2)$	$R=R_1 \odot R_2$	$D=DBLE(R_1) \odot D_2$	$C=CMPLX(R_1, 0.) \odot C_2$
D_1	$D=D_1 \odot DBLE(I_2)$	$D=D_1 \odot DBLE(R_2)$	$D=D_1 \odot D_2$	禁止
C_1	$C=C_1 \odot CMPLX(I_2, 0.)$	$C=C_1 \odot CMPLX(R_2, 0.)$	禁止	$C=C_1 \odot C_2$

[16] 算術演算子を２つ以上連続して用いることはできない
[17] 2**3**4 は、2**(3**4) と解釈される
[18] A*B/C*D は、((A*B)/C)*D と解釈される
[19] A+B-C+D は、((A+B)-C)+D と解釈される

- 冪乗の結果の型および解釈は下表の通り。ただし、I_1**I_2 で I_2 が負の場合には、1/(I_1**ABS(I_2)) に整数除算の規則を適用して解釈する。

X_1	X_2			
	I_2	R_2	D_2	C_2
I_1	I=I_1**I_2	R=REAL(I_1)**R_2	D=DBLE(I_1)**D_2	C=CMPLX(REAL(I_1, 0.)**C_2
R_1	R=R_1**I_2	R=R_1**R_2	D=DBLE(R_1)**D_2	C=CMPLX(R_1, 0.)**C_2
D_1	D=D_1**I_2	D=D_1**DBLE(R_2)	D=D_1**D_2	禁止
C_1	C=C_1**I_2	C=C_1**CMPLX(R_2, 0.)	禁止	C=C_1**C_2

整数除算

整数型の被演算子同士の除算の結果は、その絶対値が数学的な商の絶対値を超えないもっとも大きな整数値で、符号は数学的な商の符号と同じになる[20] 。

2.10.2 文字式

文字式は、文字列を表現するために用いられ、評価した結果は文字型になる。

文字演算子

文字演算子は、下表の一つだけである。連結演算 X_1//X_2 の結果は、X_1 の値の右側に X_2 の値を連結した値であり、X_1 と X_2 の長さの和を長さとする文字列。

演算子	意味
//	連結

文字式の構成規則

1. 文字定数、文字定数名、文字変数、文字配列要素、文字部分列、文字関数の引用は（単独でも）文字式
2. 文字式を括弧で括ったものも文字式
3. 2つの文字式を文字演算子で結合させたものも文字式[21]
4. 文字演算子を2つ以上含む文字式では左から右へ結合される

2.10.3 関係式

関係式は、2つ以上の算術式または文字式の値を比較するために用いられ、評価した結果は、真または偽の値を持つ論理型になる。

関係演算子

演算子	意味
.GT.	大きい (>)
.GE.	大きいか等しい (≥)
.NE.	等しくない (≠)
.EQ.	等しい (=)
.LE.	小さいか等しい (□)
.LT.	小さい (<)

関係式の値

関係式	値
e_1.GT.e_2	$e_1 > e_2$ なら真、そうでなければ偽
e_1.GE.e_2	$e_1 \geq e_2$ なら真、そうでなければ偽
e_1.NE.e_2	$e_1 \neq e_2$ なら真、そうでなければ偽
e_1.EQ.e_2	$e_1 = e_2$ なら真、そうでなければ偽
e_1.LE.e_2	e_1 □ e_2 なら真、そうでなければ偽
e_1.LT.e_2	$e_1 < e_2$ なら真、そうでなければ偽

[20] たとえば、3/5は0、7/5は1、(-7)/5は-1になる
[21] 文字演算子を2つ以上連続させることはできない

- 関係式によって、算術式と文字式の値を比較することはできない
- 算術関係式（関係式で比較される式がともに算術式であるもの）で、関係演算子を挟んだ２つの算術被演算子の型が異なる場合、関係式 $e_1 \odot e_2$ の値は、関係式 $((e_1)-(e_2)) \odot 0$ の値になる。ただし、0 は、$((e_1)-(e_2))$ と同じ型のもの。
- 倍精度実数式と複素数式との比較は不可
- 複素数式の比較では、.EQ. または.NE. のみ許される
- 文字関係式で２つの被演算子の長さが異なる場合には、短い方の右側に空白を補い、長い方の長さまで拡張して評価する
- 文字関係式での比較は文字の大小順序に基づく。e_1 の値が e_2 の値より前にあれば e_1 は e_2 より小さい。比較は両方の文字列の左端から１字ずつ行う。

2.10.4 論理式

論理式は論理演算を表現するために用い、評価した結果は真または偽の値を持つ論理型になる。
論理演算子には、下表の５種がある。

演算子	意味	優先順位
.NOT.	否定	高
.AND.	論理積	｜
.OR.	論理和	｜
.EQV.	論理等価	｜
.NEQV.	論理非等価	低

論理式の構成規則

1. 論理定数、論理定数名、論理変数、論理配列要素、論理関数の引用、関係式は論理式
2. 論理式を括弧で括ったものも論理式
3. a を論理式とするとき、.NOT.a も論理式。ただし、a は.NOT. で始まらないものとする[22]
4. 論理式を .AND., .OR., .EQV., .NEQV. で結合したものも論理式
5. .AND. が２回以上続くときは、左から右へ結合される
6. .OR. が２回以上続くときは、左から右へ結合される
7. .EQV. または.NEQV. が２回以上続くときは、左から右へ結合される

論理式の値

論理式	値
.NOT.b_1	b_1 が真ならば偽、b_1 が偽ならば真
b_1.AND.b_2	b_1, b_2 が共に真のときのみ真、それ以外は偽
b_1.OR.b_2	b_1, b_2 が共に偽のときのみ偽、それ以外は真
b_1.EQV.b_2	b_1, b_2 が共に真か共に偽のとき真、それ以外は偽
b_1.NEQV.b_2	b_1, b_2 が共に真か共に偽のとき偽、それ以外は真

[22] .NOT.(a) はつねに論理式、たとえば.NOT.(.NOT.A) は論理式

2.10.5 演算子の優先順位

式には2種類以上の演算子が含まれていてもよく、その場合の演算子の優先順位は次のとおり

算術演算子	文字演算子	関係演算子	論理演算子
高			低

2.10.6 式の評価規則

- 式の中で被演算子として引用される変数、配列要素、関数、文字部分列は、引用される時点で確定していなければならない
- 結果が数学的に定義されない算術演算[23] は、実行可能プログラムの実行時に現れてはならない
- 式の被演算子を全て評価しなくても式の値を決定できる場合、式の一部の被演算子を評価しないことがある
- 1つの文に関数の引用が2つ以上ある場合、関数を評価する順序は処理系によって異なる
- 算術式は、算術式の構成規則によって解釈されるが、評価の順序は処理系によって異なる[24]
- 関係式の評価の仕方は処理系によって異なる[25]
- 論理式の評価の順序は処理系によって異なる[26]

2.11 宣言文

2.11.1 DIMENSION文

DIMENSION $a(d)$ [,$a(d)$] ...
$a(d)$ は配列宣言子

DIMENSION文は、配列名とその次元数及び各次元の寸法を指定する。英字名 a をそのプログラム単位で配列名として使用することを宣言する。

2.11.2 型宣言文

プログラム単位の中で使用する定数名、変数名、配列名、関数名の型を宣言する文。

type v[,v]...
type は、INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL のいずれか
それぞれ整数型、実数型、倍精度実数型、複素数型、論理型に対応
処理系によっては REAL*8 のように '型*バイト数' という書き方が許されることもある
v は定数名、変数名、配列名、配列宣言子、関数名、仮手続き名

[23] ゼロによる除算、底がゼロで冪指数がゼロか負の冪乗、底が負で冪指数が実数型か倍精度実数型の冪乗など
[24] 数学的同値を保って変形が行われることがある
[25] 例えば、I, J を整変数とするとき、I.GT.J を J-I.LT.0 と評価する場合もある
[26] 例えば、論理式 b_1.OR.b_2 に対し、b_1 を先に評価して b_1 が真ならば式の評価を終え、b_1 が偽の場合 b_2 の評価に移る系もあれば、逆の順序で評価する系もある

CHARACTER [*len[,]] nam [,nam]...
nam は v[*len] または a[(d)][*len]
v は定数名、変数名、関数名、仮手続き名、a は配列名、a(d) は配列宣言子、len は文字定数、文字変数、文字配列要素、文字関数の長さ (文字の個数) で、ゼロでない符号なし整定数、正の値を持つ整定数式を括弧で括ったもの、星印を括弧で括ったもの (*) のいずれか

CHARACTER 文は定数名、変数名、配列名、関数名が文字型であることを宣言し、その長さを指定する。CHARACTER 文の直後の len は、個々に長さ指定を持つものを除く、その文中全ての要素に対する長さ指定である。長さ指定がない場合、長さは 1 になる。

2.11.3 その他の宣言文

PARAMETER (p = e [, p = e]...)
p は英字名、e は定数式

PARAMETER 文は、定数に英字名を付ける。等号の右辺の式 e の値が、代入文の割当て規則に従って p に与えられる。

COMMON [/[cb]/] nlist [[,]/[cb]/nlist] ...
cb は共通ブロック名、nlist は変数名、配列名、配列宣言子をコンマで区切って並べたもの

COMMON 文は 2 つ以上のプログラム単位間で、それぞれで使用する変数や配列の記憶場所を共用させる。COMMON 文に記述する変数や配列は複数の共通ブロック (common block) に分けることができ、名前も付けられる。

文字変数や文字配列が共通ブロックにある場合、その共通ブロックの要素は全て文字型でなければならない点に注意。

EXTERNAL proc [,proc] ...
proc は外部手続き名、仮手続き名、初期値設定副プログラム名

EXTERNAL 文は、英字名が外部手続き名あるいは仮手続き名であることを宣言する。外部手続き名や仮手続き名を他の外部手続きの実引数として用いる場合、その外部手続きを引用するプログラム単位において EXTERNAL 文による宣言が必要である。

2.12 代入文

2.12.1 算術代入文

v=e
v は整数型、実数型、倍精度実数型、複素数型の変数名または配列要素名、e は算術式

算術代入文が実行されると、式 e が評価され、下の規則に従って e が v の型に変換され、結果の値が v に代入されることによって v が確定となる。

v の型	整数型	実数型	倍精度実数型	複素数型
代入される値	INT(e)	REAL(e)	DBLE(e)	CMPLX(e)

2.12.2 論理代入文

v=b
v は論理型の変数名または配列要素名、b は論理式

論理代入文が実行されると、論理式bが評価され、bの値がvに代入されることによってvが確定となる。

2.12.3 文字代入文

v=e vは文字型の変数名、配列要素名、部分列名、eは文字式

文字代入文が実行されると、文字式eが評価され、eの値がvに代入されることによってvが確定となる。vとeの長さは異なっていてもよい。vのほうがeよりも長い場合にはvと同じ長さになるまでeの右側が空白で拡張されたのもが、短い場合にはeの左端からvと同じ長さだけ、vに代入される。

2.12.4 文番号代入文

ASSIGN s TO i sは文番号、iは整変数名

文番号代入文は、そのあとにくる割当て形GO TO文での飛び先または入出力文中の書式識別子を定義する。文番号代入文が実行されると、文番号sが整変数iに代入され、iが文番号で確定となる。ただし、文番号sは、この文と同じプログラム単位内の実行文またはFORMAT文の文番号でなければならない。

2.13 制御文

2.13.1 GO TO文

単純GO TO文

GO TO s sは同じプログラム単位中の実行文の文番号

単純GO TO文が実行されると、文番号sの文が次に実行される。

計算形GO TO文

GO TO (s [,s] ...) [,] i iは整数式、sは同じプログラム単位中の実行文の文番号

計算形GO TO文が実行されると、iが評価され、括弧内の左からi番目の文番号をもつ文が次に実行される。ただし、iは、nを文番号の並びにある文番号の個数としたとき、$1 \square i \square n$でなければならず、$i < 1$または$i > n$の場合、この文はCONTINUE文と同様に扱われる。また、括弧内の文番号の並びの中に二度以上書くことは許されている。

割当て形GO TO文

GO TO i [[,] (s [,s] ...)] iは整変数、sは同じプログラム単位中の実行文の文番号

割当て形GO TO文が実行されると、これ以前に実行された文番号代入文によってiに代入されている文番号をもつ文が次に実行される。括弧で括った文番号の並びを与えた場合、iに代入されている文番号はその並びの中の1つでなければならない。また、文番号の並びを与えない場合には、単純GO TO文と同様に働く。

2.13.2 IF 文

ブロック IF 文

```
IF (b) THEN
  [ IF ブロック ]
[ ELSE IF (b) THEN
  [ ELSE IF ブロック ]] ...
[ ELSE
  [ ELSE ブロック ]]
END IF
```
b は論理式

ブロック IF 文が実行されると *b* が評価され、その値が真の場合、IF ブロックに制御が移る。偽の場合、次の ELSE IF 文か ELSE 文か END IF 文に制御が移る。ELSE IF 文が実行されると *b* が評価され、その値が真の場合、ELSE IF ブロックに制御が移る。偽の場合、次の ELSE IF 文か ELSE 文か END IF 文に制御が移る。ELSE 文が実行されると、その ELSE ブロックに制御が移る[27] 。END IF 文はブロック IF 文に対応し、その影響の及ぶ範囲の終わりを示す。

論理 IF 文

```
IF (b) st
```
b は論理式、*st* は実行文（ただし、DO 文、ブロック IF 文、ELSE IF 文、ELSE 文、END IF 文、END 文、論理 IF 文を除く）。

論理 IF 文が実行されると *b* が評価され、その値が真ならば文 *st* が実行される。偽の場合、次の行へ制御が移る。

算術 IF 文

```
IF (e) s1,s2,s3
```
e は算術式、s_1, s_2, s_3 は同じプログラム単位内の実行文の文番号。

算術 IF 文が実行されると *e* が評価され、$e<0$ ならば文番号 s_1 の文、$e=0$ ならば文番号 s_2 の文、$e>0$ ならば文番号 s_3 の文が次に実行される[28] 。

2.13.3 DO 文

```
DO s [,] i=e1,e2[,e3]
```
s は同じプログラム単位内で当該 DO 文以降にある実行文の文番号、*i* は整数型、実数型、倍精度実数型の変数名、e_1, e_2, e_3 は整数型、実数型、倍精度実数型の式。

DO 文は、それに続く最初の文から端末文（文番号 *s* の文）までのすべての実行文（DO ループ）を繰り返し実行することを指示する。DO ループの範囲内に DO 文がある場合、その DO 文によって指定される DO ループの範囲は、外側の DO ループの範囲内に完全に含まれていなければならない。ただし、2 つ以上の DO ループの末端文が同じであることは許される。IF ブロック、ELSE IF ブロック、ELSE ブロック内に DO 文がある場合、その DO ループの範囲は IF ブロック、ELSE IF ブロック、ELSE ブロック内に含まれていなければならない。また、DO ループの範囲内にブロック IF 文がある場合、対応する END IF 文は同じ範囲に含まれていなければならない。

端末文は、実行文でなければならないが、GO TO 文、IF 文、RETURN 文、STOP 文、END 文、DO 文を端末

[27] IF ブロック、ELSE IF ブロック、ELSE ブロックの外からその中へ制御を移してはならない
[28] 1 つの算術 IF 文中に同じ文番号が 2 回以上現れてもよい

文とすることはできない[29] 。

DO文の実行

1. e_1, e_2, e_3 が評価され、初期値パラメター m_1、終値パラメター m_2、増分パラメター m_3 が決まる。このとき、必要に応じてDO変数の型に変換される。e_3 がない場合、$m_3 = 1$ となる。
2. DO変数に初期値パラメター m_1 の値が代入される
3. 繰返し数が MAX(INT((m2-m1+m3)/m3),0) によって決められる[30]

以上の3段階が行われ、DO文の実行が終わるとループ制御処理に移る。

ループ制御処理

繰返し数を調べ、その値がゼロでない場合、そのDOループの範囲の最初の実行文の実行に移る。繰返し数の値がゼロの場合、DOループを休止状態にする。その結果、このDOループの端末文を共有しているDOループがすべて休止状態になれば、この端末文に続く実行文に制御が移る。端末文を共有しているDOループで活動状態のものがあれば増分処理に移る。

DOループの範囲の実行

端末文に達するまでDOループの範囲にある文が実行される。DO変数はDOループの範囲の実行中に再定義あるいは不定にしてはならない。端末文が実行されると、増分処理に移る。

増分処理

1. 一番最後に実行したDO文によって活動状態にあるDOループのDO変数、繰返し数、増分パラメターを選ぶ
2. DO変数の値に増分パラメター m_3 の値を加える
3. 繰返し数を1減らす
4. このDOループのループ制御処理に移る

2.13.4 CONTINUE文

```
CONTINUE
```

CONTINUE文は、実行時には何もしない。通常DOの端末文として用いられる。また、単に文番号を設定するためにも用いられる。

2.13.5 STOP文

```
STOP [n]
n は文字定数あるいは5桁以内の数字列
```

STOP文は、実行可能プログラムの実行を終了する。n が指定された場合、n と出力して実行可能プログラムの実行を終了する。

---

[29] DOループがループを形成しなくなってしまうため

[30] $m_3 \neq 0$ でなければならない。また、$m_1 > m_2$ かつ $m_3 > 0$ あるいは $m_1 < m_2$ かつ $m_3 < 0$ の場合、繰返し数は0。

2.13.6 END 文

```
END
```

END 文は、プログラム単位の終わりを示す。主プログラムで END 文が実行されると、その実行可能プログラムが終了する。関数副プログラムやサブルーチン副プログラムで END 文が実行されると、RETURN 文と同様の効果があり、その副プログラムを引用したプログラム単位に制御が戻る。

2.14 入出力文

入力文 外部媒体または内部ファイルから内部記憶へデータを転送する

出力文 内部記憶から外部媒体または内部ファイルへデータを転送する

記録 (record) 値の列または文字の列

- **書式付き記録** 書式付き入出力文によって読み書きできる記録。処理系で表現可能な任意の文字の列で構成される。記録内の文字の個数をその書式付き記録の長さという[31] 。
- **書式なし記録** 書式なし入出力文によって読み書きできる記録。処理系によって定められた形をもつ値の列で構成される。
- **ファイル終了記録** 1 つのファイルの最後の記録で、ENDFILE 文によって書かれる

ファイル 記録の列[32]

- **外部ファイル** 外部記憶装置 (磁気ディスクなど) 上のファイルで、プログラム実行後も保存しておくことができる
- **内部ファイル** 内部記憶にプログラムの実行中にだけ存在するファイル

ファイルの位置

- **始点** ファイルの最初の記録の直前の位置
- **終点** ファイルの最後の記録の直後の位置

ファイルのアクセス

- **逐次アクセス** 記録の順序は、それが書かれた順序 (直接アクセスが許されていないファイルの場合)、または直接アクセスに対して指定された記録番号順 (直接アクセスも許されているファイルの場合)。ファイルの記録は、すべて書式付けられているか、またはすべて書式付けられていないかのいずれか。ただし、ファイルの最後の記録はファイル終了記録でもよい。
- **直接アクセス** 記録の順序は記録番号順。ファイルの記録は、すべて書式付けられているか、またはすべて書式付けられていないかのいずれか。

内部ファイル 内部記憶間でデータを転送したり変換したりするために用いることができる。ファイルは、文字変数か文字配列要素か文字配列か文字部分列。内部ファイルの記録である変数、配列要素、部分列は、その記録を書くことによって確定になる。記録に書かれた文字数が記録の長さより少ない場合、残りの部分には空白が埋められる。逐次アクセス書式付き入出力文でのみ読み書きできる。

---

[31] 長さの限度は処理系および外部媒体に依存する。長さはゼロでもよい。
[32] ファイルはデータセットと呼ばれることもある

装置 ファイルを参照するための手段。装置をファイルに接続することによって、ファイルを参照できる。キーボードからの入力に対する標準の入力ファイルには装置5、ディスプレイ画面への出力に対する標準の出力ファイルには装置6があらかじめ接続されている。その他のファイルに付いては、装置をコマンドまたはジョブ制御言語によって事前に接続しておくかOPEN文によって接続する必要がある。

2.14.1 指定子

装置指定子

[UNIT=]u u は外部装置識別子または内部ファイル識別子

外部装置識別子は外部ファイルを参照するためのもので、整数式 $i(\geq 0)$ または書式付き逐次アクセスをするものとして事前に接続されている処理系で定めた特定の外部装置を指す星印 '*'。

内部ファイル識別子は内部ファイルを参照するためのもので、文字変数名、文字配列名、文字配列要素名、部分列名のいずれか。

書式指定子

[FMT=]f f は書式識別子

書式識別子はデータを転送する際に用いる書式を識別するためのもので、次のいずれか。

1. 同じプログラム単位内のFORMAT文の文番号
2. 同じプログラム単位内のFORMAT文の文番号が代入されている整変数名
3. 文字配列名
4. 文字式
5. 並びによる書式を指定する星印 '*'

記録指定子

REC=rn $rn(>0)$ は整数式

rn により、直接アクセスをするものとして接続されるファイルの読み書きすべき記録の番号を指定する。

入力状態指定子

IOSTAT=ios ios は整数型の変数または配列要素

この指定子をもつ入出力文が実行されると、

$$ios = \begin{cases} 0 & \text{(誤り条件とファイル終了条件のいずれも検出されなかった場合)} \\ \text{正の整数} & \text{(誤り条件が検出された場合)} \\ \text{負の整数} & \text{(ファイル終了条件が検出され、かつ誤り条件が検出されなかった場合)} \end{cases}$$

で値が確定[33]。

[33] 整数値は処理系によって異なる

誤り指定子

ERR=s
s は同じプログラム単位内の実行文の文番号

この指定子をもつ入出力文の実行中に誤り条件が検出されると、その入出力文の実行が中止され、その入出力文で指定されたファイルの位置が不定となり、文番号 s の文から実行が続けられる。

ファイル終了指定子

END=s
s は同じプログラム単位内の実行文の文番号

この指定子をもつ READ 文の実行中にファイル終了条件が検出されかつ誤り条件が検出されなかった場合、READ 文の実行は中止され、文番号 s の文から実行が続けられる。ファイル終了条件は、逐次アクセスでファイルを読んでいる最中にファイル終了記録が検出された場合、および内部ファイルの終わりをこえて記録を読もうとした場合に発生する。

制御情報並び

装置指定子、書式指定子、記録指定子、入力状態指定子、誤り指定子、ファイル終了指定子の各項目からなる並びを制御情報並びという。制御情報並びには同じ種類の指定子を 2 つ以上書くことはできない。装置指定子は必ずなければならないが、それ以外の指定子はなくてもよく、また指定子を書く順序は任意である。ただし、'UNIT=' を省略する場合、装置指定子 u は最初に書かなければならず、'FMT=' を省略する場合、1 番目に装置指定子 f、2 番目に書式指定子 f を書かなければならない。

書式付き入出力文 書式指定子がある入出力文

書式なし入出力文 書式指定子がない入出力文

並びによる入出力文 書式識別子が星印 '*' の入出力文

直接アクセス入出力文 記録指定子がある入出力文

逐次アクセス入出力文 記録指定子がない入出力文

入出力並び

データ転送入出力文によって値が転送される項目を指定する並びを入出力並びという。

入出力並び項目

変数名、配列名、部分列名、配列名のいずれか。出力並び項目については、さらに任意の式も許される。

DO 形並び

($dlist$,i=e_1,e_2,[e_3])
i は整数型、実数型、倍精度実数型の変数名、e_1, e_2, e_3 は整数型、実数型、倍精度実数型の式、$dlist$ は入出力並び

繰返し数および DO 変数の値 i は、DO ループと同様 e_1, e_2, e_3 から決められる。

2.14.2 READ文、WRITE文、PRINT文

READ(*clist*)[*iolist*]
READ *f*[,*iolist*]
WRITE(*clist*)[*iolist*]
PRINT *f*[,*iolist*]
clist は制御情報並び、*f* は書式識別子、*iolist* は入出力並び

READ文が実行されるとき、入力並びが指定されていればそこで指定された項目にファイルから値が転送される。制御情報並びを含まないREAD文は、制御並びを含むREAD文で装置指定子を星印'*'としたときと同じ装置を指定したことになる。

WRITE文またはPRINT文が実行されると、出力並びおよび書式仕様で指定した項目からファイルへ値が転送される。PRINT文はWRITE文で装置指定子を星印'*'にしたときと同じ装置を指定したことになる。

2.14.3 補助入出力文

OPEN(*olist*)
olist は指定子 [UNIT=]*u* IOSTAT=*ios* ERR=*s* FILE=*fin* STATUS=*sta* ACCESS=*acc* FORM=*fm* RECL=*rl* BLANK=*blnk* の並び

OPEN文はすでに存在するファイルを装置に接続する、事前に接続されているファイルを生成する、ファイルを生成し装置に接続する、ファイルと装置との間の接続に用いた指定子を変更する、ということを行う。*olist* には外部装置指定子は必ずなければならないが、それ以外の指定子はなくてもよい。

CLOSE(*cllist*)
cllist は指定子 [UNIT=]*u* IOSTAT=*ios* ERR=*s* STATUS=*sta* の並び

CLOSE文は特定のファイルと装置との接続を解除する。*cllist* には外部装置指定子は必ずなければならないが、それ以外の指定子はなくてもよい。

2.15 書式仕様

2.15.1 編集記述子

記録の形を指定し、記録中の文字列とデータの内部表現との間の編集を指示する。

反復可能編集記述子

I*w*　I*w*.*m*	G*w*.*d*　G*w*.*d*E*e*
F*w*.*d*	L*w*
E*w*.*d*　E*w*.*d*E*e*	A　A*w*
D*w*.*d*	

I, F, E, D, G, L, Aは編集方法、*w* および *e* はゼロでない符号なし整定数、*d* および *m* は符号なし整定数

反復不能編集記述子

'$h_1h_2\cdots h_n$'　*n*H$h_1h_2\cdots h_n$	:
T*c*　TL*c*　TR*c*	S　SP　SS
*n*X	*k*P
/	BN　BZ

', H, T, TL, TR, X, /, :, S, SP, SS, P, BN, BZは編集方法、*h* は処理系で表現可能な1文字、*n* および *c* はゼロでない符号なし整定数、*k* は整定数

2.15.2 編集

整数の編集

整数型データの編集には Iw および Iw.m 形編集記述子を用いる。ここで、w は欄の幅を表し、$w \geq m$ でなければならない。

入力

- 入力欄中のデータは整定数の形をしていなければならない
- Iw.m 形編集記述子を用いても Iw 形編集記述子を用いても編集は同じ

出力

- Iw 形編集記述子は内部データの値の絶対値を、頭にゼロのつかない符号なし整定数の形に編集する。内部データの値が負の場合、その前に負符号が付加される。
- Iw.m 形編集記述子は、基本的には Iw 形編集記述子と同じであるが、符号なし整定数が少なくとも m 桁からなるように、必要ならば頭に 0 を付ける。m がゼロでかつ内部データの値がゼロの時は欄は空白文字だけで構成される。

実数および倍精度実数の編集

実数型および倍精度実数型データの編集には Fw.d、Ew.d、Ew.dEe、Dw.d、Gw.d、Gw.dEe 形編集記述子を用いる。ここで、w は欄の幅、d は小数部の桁数、e は指数部の桁数を表す[34] 。

入力

- 入力データは数字列で構成し、先頭に符号を付けてもよく、小数点を含んでいてもよい。小数点がない場合、数字列の右側 d 桁が小数部と見なされ、必要に応じて頭にゼロがあるものとみなされる[35]
- 数字列は処理系がそのデータの値を近似的に表現できる桁数より長くてもよい
- 入力データに指数部をつけてもよい。指数部は次のような表現が可能： 1.2×10^{3} に対して 1.2E+03, 1.2E+3, 1.2E03, 1.2E3, 1.2D+03, 1.2D+3, 1.2D03, 1.2D3, 1.2+03, 1.2+3、1.2×10^{-3} に対して 1.2E-03, 1.2E-3, 1.2D-03, 1.2D-3, 1.2-03, 1.2-3。

出力

- F 形編集の場合、小数点を含む数字列で出力する。値の絶対値が 1 より小さい場合、小数点の左隣にゼロが付加される。
- F, E, D 形編集の場合、結果は小数点以下 d 桁に丸めたものが出力される。
- E, D 形編集の場合、出力欄は [±][0].$x_1x_2\cdots x_d$ exp になる。ここで、± は符号、$x_1x_2\cdots x_d$ は出力するデータの値の仮数部を d 桁に丸めたときの有効数字。exp は 10 進指数でその形は、Ew.d の場合 E±z_1z_2、Ew.dEe の場合 E±$z_1z_2\cdots z_e$、Dw.d の場合 D±z_1z_2 となる。符号等の欄を考慮して、$w > d+7$ であることが必要。
- G 形編集の場合（省略）

[34] 入力では e は効果をもたない
[35] データの小数点の位置と編集記述子で指定した小数点の位置とが異なっている場合、データの小数点の位置が採用される

複素数の編集

複素数型データは一対の実数型データからなるため、編集には引き続く2つのF, E, G 形編集記述子を用いる。はじめの編集記述子が実部、2番目の編集記述子が虚部に対応する[36] 。

数値の入出力に関する規則

入力

- 入力データの頭の空白は意味をもたない。全てが空白の欄はゼロとみなす。
- 入力データが正の数値の場合正符号は付けても付けなくてもよい。負の数値の場合は必ず先頭に負符号を付けなければならない。

出力

- 出力される数値は、すべて指定した欄に右寄せに置かれる。出力される字数が指定した欄の幅より小さい場合、余った分だけ数値の前に空白がとられる。
- 編集によってできる文字の個数が欄の幅を超える場合、その欄は星印 '*' で埋められる。
- 負の値に対しては必ず数値の前に負符号がつく。正またはゼロの値に対しては、S, SP, SS 形編集記述子または処理系が定めた制御によって正符号がつくことがある。

文字の編集

文字型データの編集には A 形編集記述子を用いる。また、指定した文字の列を出力するためのアポストロフィ編集記述子 ''', H 形編集記述子を用いることもできる。

アポストロフィ編集

空白を含む文字の列を出力できる。欄の幅はアポストロフィで括った文字の個数になる。アポストロフィ ''' を出力する場合、1つのアポストロフィに対して連続する2つのアポストロフィ '''' を用いる。

アポストロフィ編集と同様の編集を H 形編集記述子で行うこともできる。この場合、出力したい n 個の文字の列の前に nH と書く。

A 形編集

欄の幅 w を指定すると欄は w 個の文字から構成される。指定しない場合、欄の文字の個数は対応する文字型の入出力並び項目の長さになる。

入出力並び項目の長さ len に対し、入力の場合指定した欄の幅 w が len 以上であれば入力欄の右端の len 個の文字がとられ、小さければ w 個の文字の後に $len-w$ 個の空白が付加される。出力の場合、欄の幅 w が len より大きければ $w-len$ 個の空白が左側に取られ、続いて len 個の文字が出力される。小さい場合には、左側から w 個の文字が出力される。

論理型の編集

論理型データの編集には Lw 形編集記述子を用いる。

入力欄の w 桁の欄内の最初の空白以外の文字が T の場合には .TRUE., F の場合には .FALSE. が対応する入力並び項目に与えられる。T, F の前に小数点があってもよく、T, F の後に文字が続いていてもよい。

[36] 2つの編集記述子は異なっていてもよい

出力の場合、$w-1$ 個の空白が取られ、次に、対応する出力並び項目の真偽に応じて T または F が出力される。

位置付け編集

転送する文字の位置を指定するには、T, TL, TR, X 形編集記述子を用いる。

Tc 編集記述子は、次の文字の転送を記録の c 番目の文字位置から行うことを指示する。

TLc 編集記述子は、次の文字の転送を現在の位置から c 文字戻した文字位置から行うことを指示する。ただし、現在の位置が c 以下の場合、現在の記録の 1 番目の位置から転送を行う。

TRc 編集記述子は、次の文字の転送を現在の位置から c 文字進めた文字位置から行うことを指示する。

nX 編集記述子は、TRn と同じ意味である。

T, TR, X 形編集記述子によって指定した位置またはそれ以降に文字を転送する場合、飛び越した部分で以前に文字を入れなかった位置には空白が埋め込まれる。

2.15.3 正符号の制御

正の数値を出力する場合に正符号を付けるかどうかは、SP, SS, S 形編集記述子で指定する。

SP 形編集記述子は、これ以降の出力に対して正符号を付けることを指示する。

SS 形編集記述子は、これ以降の出力に対して正符号を付けないことを指示する。

S 形編集記述子は、正符号の出力を処理系に任せることを指示する。

特に指定しない場合、正符号出力の扱いは処理系によって異なる。

数値入力欄中の空白の解釈

数値入力欄中の空白の解釈は、BN, BZ 形編集記述子で指定する。

BN 形編集記述子は、これ以降の数値入力欄の空白を全て無視することを指示する。

BZ 形編集記述子は、これ以降の数値入力欄の空白を全てゼロとして扱うことを指示する。

桁移動数

F, E, D, G 形編集記述子に対しては、P 形編集記述子 kP を用いて桁移動数を指定できる。

- F 形編集記述子による出力および欄内に指数がないものの入力に対する F, E, D, G 形編集記述子に対しては、外部表現 = 内部表現 $\times 10^k$ となる。
- 欄内に指数をもつデータの入力には影響しない。
- E, D 形編集記述子による出力に対しては、出力する量の仮数部（基本実定数）が 10^k 倍され、指数部は k だけ減らされる。したがって、この場合にはもとの数値と大きさにおいて変わりがない。

文字の入出力の制御

入出力並びに項目が残っていなければ書式制御を終了させたいときには、コロン編集記述子 ':' を用いることができる[37]。

[37] 入出力並びに項目が残っているときには、何の影響も与えない

複数の記録をまとめて指定する

斜線編集記述子 '/' により、いくつかの記録の書式をまとめて指定する。斜線は記録の区切りを示すものであり、入力の場合には新たに次の記録を読み込むことを、出力の場合にはそれまでに作られた記録を書き出すことを指定する。

2.15.4 書式仕様の形

([*flist*])
flist は項目 [*r*]*ed*, *ned*, [*r*]*fs* の並び。*ed* は反復可能編集記述子、*ned* は反復不能編集記述子、*fs* は空でない *flist* をもつ書式仕様、*r* はゼロでない符号なし整定数

flist 中の並び項目を区切るカンマ ',' は、P 形編集記述子とその直後の F, E, D, G 形編集記述子との間、斜線編集記述子 '/' の前後、コロン編集記述子 ':' の前後の場合、省略可能。

書式仕様の与え方

FORMAT 文による書式仕様

FORMAT *fs*
fs は書式仕様

FORMAT 文に文番号を付け、その番号を指定することで書式仕様を与えられる。

文字による書式仕様

文字配列、文字変数、その他の文字式として書式仕様を与えられる。この場合、その文字配列名、文字変数名、その他の文字式を書式付き入出力文の書式識別子として指定する。文字配列で与える場合、添字の値に従ってその配列要素を全て連結したもので与えることになる。

並びによる書式

書式識別子が星印 '*' の場合に採られる書式を並びによる書式という。詳細は省略。

2.16 主プログラム、関数、サブルーチン

2.16.1 主プログラム

実行可能プログラムの実行が開始されるプログラム単位を主プログラムという。実行可能プログラムの実行は主プログラムの最初の実行文から開始される。

PROGRAM *prog*
prog はプログラム名

PROGRAM 文により、そのプログラム単位が主プログラムであることを指示する。

2.16.2 関数の引用

func([*a*[,*a*]···])
func は関数または仮手続きの英字名、*a* は実引数

式中の関数の引用が実行されると、関数 *func* の評価が開始される。引用した関数 *func* から制御が戻ると、その関数の引用の実行が終了し、それを引用している式でその関数の値が使用可能になる。

2.16.3 組み込み関数

FORTRAN には予め組込み関数として、型変換や基本的な数学関数が用意されている。詳しくはマニュアルを参照のこと。結果が数学的に定義されなかったり、処理系の数値の範囲を超える結果を与えるような引数に対しては、関数の値は不定になることに注意。

2.16.4 文関数

文関数は、算術代入式、論理代入式、文字代入式に似た、ただ 1 つの文を用いて定義する関数。
関数値は関数名に与えられる。したがって関数の型は関数名の型に従う。
文関数定義文は、そのプログラム単位の最初の実行文よりも前で、宣言文よりも後に置く。

文関数定義文
func([*d*[,*d*]···])=*e*
func は文関数名、*d* は仮引数、*e* は *func* の型が論理型ならば論理式、文字型ならば文字式、それ以外の場合は算術式

仮引数 *d* は変数名で、型、個数、順序を規定するだけであり、その変数名の有効範囲はその文関数定義文。同じ英字名を 1 つの文関数の仮引数の並びの中で二度以上使うことはできない。仮引数名は別の文関数定義文で、同じ型の他の仮引数として用いることはできる。また、その名前をそのプログラム単位中で同じ型の他の変数名として用いることもできる。

式 *e* の中には、仮引数以外に定数、定数名、変数、配列要素、定義済みの文関数、組込み関数、外部関数、仮手続きを用いることができる。

文字型の文関数、文関数の文字型の仮引数の長さ指定は整定数式でなければならない。

文関数は、定義をしたプログラム単位中でのみ引用できる。文関数を引用するには、適当な実引数をつけて関数名を式の中に書けばよい。実引数には式を用いるが、仮引数と個数、順序、型が一致していなければならない。

2.16.5 関数副プログラム

関数副プログラムは、主プログラムまたは他の副プログラムで引用して使用する外部関数を定義する。

関数副プログラムは、その結果を副プログラム名に与えるため、関数の型は副プログラム名で指定または宣言するか、あるいは FUNCTION 文の FUNCTION の語の前に INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, CHARACTER [*len*] をつける。文関数定義文は、そのプログラム単位の最初の実行文よりも前で、宣言文よりも後に置く。

関数副プログラムの定義は、FUNCTION 文で行う。

[*typ*] FUNCTION *func*([*d*[,*d*]···])
typ は関数の型宣言で INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, CHARACTER [*len*] のいずれか、*len* は文字関数の結果の長さ指定で省略時は 1、*func* は関数副プログラム名、*d* は仮引数で変数名、配列名、仮手続き名

関数副プログラムは、RETURN 文または END 文によりその副プログラムを引用したプログラムへ制御を戻さなければならない。

結果の転送は副プログラム名またはそれと結合している同じ型の入口名を媒体として行われるため、それらの英字名のいずれかは変数名として副プログラム中で少なくとも 1 度代入文の左辺に現れるかあるいは入力文の入力並びの中に現れるかなどにより、副プログラムの実行中に値が確定されなければならない。

副プログラム中では、仮引数を通常の変数、配列、手続きとして使用するため、仮引数の英字名は型を宣言する必要がある。仮引数が配列名の場合、副プログラム中で型宣言文または DIMENSION 文により宣言しなければならない。

関数副プログラムを引用するには、適当な実引数をつけて副プログラム名を式の中に書けばよい。実引数は仮引数と個数、順序、型が一致していなければならない。実引数には、定数、定数名、変数名、配列要素名、その他の式、配列名、組込み関数名、外部手続き名、仮手続き名を用いることができる[38] 。

2.16.6 サブルーチン副プログラム

サブルーチン副プログラムはサブルーチンを定義するもので、関数副プログラムと類似しているが、関数副プログラムは式中で1つの値を求めるのに対し、サブルーチン副プログラムは複数の値を求める場合にも使用できるほか、複雑な入出力などをまとめて処理する場合にも使用できるという違いがある。

サブルーチン副プログラムの結果は、引数または共通ブロックを通して転送されるため、結果の型は個々の変数、配列の型に従う。よって、サブルーチン副プログラムの名前は、他のプログラムで引用できるために一意でなければならないが、型とは無関係である。

サブルーチン副プログラムの定義は、SUBROUTINE 文で行う。

SUBROUTINE *sub*[([*d*[,*d*]···])]
sub はサブルーチン名、*d* は仮引数で変数名、配列名、仮手続き名、星印 '*' のいずれか

サブルーチン副プログラムは、RETURN 文または END 文によりその副プログラムを引用したプログラムへ制御を戻さなければならない。

サブルーチン副プログラムの結果は、引数を媒体として転送される。

サブルーチン副プログラム中では、仮引数を通常の変数、配列、手続きとして使用するため、仮引数の英字名は型を宣言する必要がある。仮引数が配列名の場合、副プログラム中で型宣言文または DIMENSION 文により宣言しなければならない。

サブルーチン副プログラムの引用は、CALL 文で行う。

CALL *sub*[([*d*[,*d*]···])]
sub はサブルーチン名または仮手続き名、*d* は実引数

指定したサブルーチンの実行が終了すると、CALL 文に続く実行文に制御が移る。CALL 文の実引数は仮引数と個数、順序、型が一致していなければならない。実引数には、定数、定数名、変数名、配列要素名、その他の式、配列名、外部手続き名、仮手続き名、選択戻り指定子 '*s*' を用いることができる。ここで、*s* は CALL 文と同じプログラム単位内の実行文に付けた行番号。

2.16.7 ENTRY 文

ENTRY *en*[([*d*[,*d*]···])]
en は関数副プログラムまたはサブルーチン副プログラムの入口の英字名、*d* は仮引数で変数名、配列名、仮手続き名、星印 '*' のいずれか

ENTRY 文により、関数副プログラムまたはサブルーチン副プログラムの特定の実行文から始まる手続きの引用が可能になる。入口名 *en* は、サブルーチン副プログラムの ENTRY 文の場合にはサブルーチン名、関数副プログラム内の ENTRY 文の場合には外部関数名である。

入口名 *en* を用いて手続きを引用すると、入口名 *en* をもつ ENTRY 文に続く最初の実行文から手続きの実行が開始される。ENTRY 文の仮引数の個数、順序、型、名前は、その副プログラムの FUNCTION 文、SUBROUTINE 文、他の ENTRY 文の仮引数と異なっていてもよい。

[38] 組込み関数名を実引数として用いる場合、INTRINSIC 文でその名前を指定しなければならない

2.16.8 RETURN 文

RETURN [e]
e は整数式

RETURN 文は引用しているプログラム単位に制御を戻す。RETURN 文は関数副プログラムまたはサブルーチン副プログラムでしか用いることはできない。関数副プログラムでは e を指定することはできない。

選択戻り

現在引用されている名前をもつ SUBROUTINE 文または ENTRY 文中の星印 '*' の個数を n とするとき、$1 \square e \square n$ であれば e の値は仮引数の並びにある e 番目の星印を識別し、それと結合している CALL 文中の選択戻り指定子で指定した文へ制御が戻る。e がない、あるいは $e < 1$ または $n < e$ の場合、その副プログラムを引用した CALL 文の次の文に戻る。

1つの副プログラムに複数の RETURN 文が存在してもよい。

2.17 Fortran90

Fortran90 で追加・拡張された部分は多数あるが、その中でも特に知っておくとよいと思われるいくつかの事項について記述しておく。

2.17.1 宣言文

type[,*attribute*,···]::*name*[,···]
type は変数のデータ型で INTEGER, REAL, COMPLEX, LOGICAL, CHARACTER のいずれか[a] 、*attribute* は変数に付加する属性で、PARAMETER (名前付き定数)、DIMENSION(*d*[,*d*,···]] (配列)、ALLOCATABLE (割り付け可能)、SAVE (状態保存)、INTENT(*io*) (授受特性)、OPTIONAL (省略可能)、POINTER (ポインタ)、TARGET (ターゲット)、INTRINSIC (組込み手続き名の宣言)、EXTERNAL (外部手続き名の宣言)、*name* は変数名、*d* は配列の寸法宣言子、*io* は授受特性で IN, OUT, INOUT のいずれか

[a] もう少し正確にいうと、同時に KIND パラメターを指定することができる

2.17.2 DO ループ

[*label*:] DO $i = e_1, e_2[, e_3]$
    DO ループ
END DO [*label*]
label はループ名、i は整数型、実数型、倍精度実数型の変数名、e_1, e_2, e_3 は整数型、実数型、倍精度実数型の式のいずれか

ループ制御

CYCLE [*label*] (*label* のループについて、増分処理を行い次のループ制御処理に移る)
EXIT [*label*] (*label* のループを終了し、次の実行文へ制御を移す)

2.17.3 割り付け配列

Fortran90 では、実行時に任意の大きさの配列を動的に割り付けて使用できる。割り付け配列の宣言は、型宣言文で ALLOCATABLE 属性を指定し、DIMENSION 属性の寸法宣言子をコロン ':' として次元数の

み指定する。

配列の割り付け、解放は ALLOCATE 文、DEALLOCATE 文で行う。

ALLOCATE (*al*[,STAT=*stat*])
DEALLOCATE (*ao*[,STAT=*stat*])
al は割り付け指定並び *obj*[([*lb*:]*ub*)]、*obj* は割り付け配列またはポインタ、*lb* は配列の下限、*ub* は配列の上限で整数式、下限の暗黙の値は 1、*stat* は状態変数で割付/解放が成功すればゼロ、失敗すれば正の値で確定、*ao* は割り付けられた割り付け配列またはポインタを指定した ALLOCATE 文で割り付けられた指示先全体と結合しているポインタ

第3章　磁気流体（MHD）シミュレーション入門

3.1　はじめに

プラズマは、プラズマの構成粒子である電子やイオンの運動などに関連したマクロスケールの物理から、それらが集団として動くことにより生まれる流体の運動や熱の輸送などマクロスケールの物理までの様々な時空間スケールの物理が混在している複雑系である。例えば、典型的な大型磁場閉じ込め装置における物理パラメータを用いると、閉じ込め空間スケール $a \approx 0.5\mathrm{m}$、イオンの旋回半径 $\rho_i \approx 3 \cdot 10^{-3}\mathrm{m}$、電子のデバイ長 $\lambda_d \approx 7 \cdot 10^{-5}\mathrm{m}$、また時間変化のスケールとしては、二体粒子間衝突の振動数 $\nu_{ei} \approx 5 \cdot 10^{3}\mathrm{s}^{-1}$、イオン旋回振動数 $\omega_{ci} \approx 3 \cdot 10^{8}\mathrm{s}^{-1}$、電子プラズマ振動数 $\omega_{pe} \approx 6 \cdot 10^{11}\mathrm{s}^{-1}$、等である。このような様々なスケールの混在する複雑プラズマを記述する方法として、目的とする現象の時間スケール・空間スケールに応じて、粒子の特徴的な時間・空間スケールの現象を記述する粒子的記述法、閉じ込めスケールでゆっくり起こる現象を記述するための流体的記述法、及び、その中間的又は複合的記述法が開発されてきた。以下で解説する磁気流体シミュレーション手法は、流体的記述法に基づいたものであり、イオンの運動を特徴づける時空間スケールより大きくゆっくりした時空間スケールの現象を記述している（空間スケール $L \gg \rho_i$，時間スケール $\tau \gg \omega_{ci}^{-1}$ ）。

3.2　基礎方程式

まず、流体的モデルに基づくプラズマの基礎方程式である、磁気流体（MHD）方程式を示す。

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0 \tag{3.1}$$

$$\rho \frac{d \boldsymbol{v}}{d t} = \boldsymbol{j} \times \boldsymbol{B} - \nabla p \tag{3.2}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times \boldsymbol{E} \tag{3.3}$$

$$\boldsymbol{E} = -\boldsymbol{v} \times \boldsymbol{B} + \eta \boldsymbol{j} \tag{3.4}$$

$$\boldsymbol{j} = \frac{1}{\mu_0} \nabla \times \boldsymbol{B} \tag{3.5}$$

$$\frac{\partial p}{\partial t} + \nabla \cdot (p \boldsymbol{v}) = (\gamma - 1)(-p \nabla \cdot \boldsymbol{v} + \eta j^2) \tag{3.6}$$

ここで、$\boldsymbol{B}$, $\boldsymbol{E}$, $\boldsymbol{j}$, $\boldsymbol{v}$, ρ, p, η および γ は、それぞれ磁場、電場、電流、プラズマ速度、密度、圧力、電気抵抗と断熱定数である。ここでは、粘性項や熱伝導効果などの枝葉を落とした最も単純な基礎方程式を示しているが、そのような様々な効果が入ってきても、これから先の話は同様である。

MHDシミュレーションとは、本質的には、これらの時間発展方程式を与えられた境界条件と初期条件の下で解くことであって、他の研究分野であっても、時間発展方程式が与えられている限り、ここで紹介するシミュレーション手法はそのまま適用できる。

通常、解きたい空間を多くの格子点に分割して、各点での物理量の時間発展を追跡するという形で、シミュレーションを実行する。格子点の分け方は、典型的には、デカルト座標系、円筒座標系、球座標系等の、採用した直角座標系に対応した形をとるが、時には、物理モデルに応じて不均一な分け方をとる場合

もある。ここでは、通常の直角座標系で話を進める。

空間微分を求める手法としては、差分法とスペクトル法の大きく分けて2つの方法がある。ただ、スペクトル法は、複雑な境界条件（例えば、境界条件が方程式で与えられるなど）には、適用が非常に困難であること、分散メモリ型のベクトルコンピュータには不向きであることから、一般的には差分法がよく用いられている。

方程式 (2)、(3)、(6) にそれぞれ左から $\boldsymbol{B}/\mu_0$、 $\boldsymbol{v}$、$1/\gamma-1$ をかけて空間積分し、方程式 (1)、(4)、(5) を用いると、

$$\frac{\partial K}{\partial t} = -\int \frac{1}{2}\rho v^2 \boldsymbol{v}\cdot d\boldsymbol{S} + \int \boldsymbol{J}\times\boldsymbol{B}\cdot \boldsymbol{v} dV - \int \nabla p\cdot \boldsymbol{v} dV \tag{3.7}$$

$$\frac{\partial W}{\partial t} = -\int \frac{\boldsymbol{E}\times\boldsymbol{B}}{\mu_0}\cdot d\boldsymbol{S} - \int \boldsymbol{J}\times\boldsymbol{B}\cdot \boldsymbol{v} dV - \int \eta J^2 dV \tag{3.8}$$

$$\frac{\partial T}{\partial t} = -\int \frac{\gamma}{\gamma-1} p\boldsymbol{v}\cdot d\boldsymbol{S} + \int \nabla p\cdot \boldsymbol{v} dV + \int \eta J^2 dV \tag{3.9}$$

となる。ここで、$K=\int \frac{1}{2}\rho v^2 dV$ 、$W=\int \frac{B^2}{2\mu_0} dV$ 、$T=\int \frac{p}{\gamma-1} dV$ で、それぞれ運動エネルギー、磁場エネルギー、熱エネルギーを表す。一目でわかるように、これらを全て足すと右辺に残るのは境界面からの寄与だけで、左辺が全エネルギーとなって、境界からのエネルギーの出入りによって全エネルギーが増減するという方程式になる。磁気流体プラズマにおいては、これら3つのエネルギーの間のやりとりと境界におけるエネルギーの授受を通して、現象が時間発展していく。注意するべき点は、3つのエネルギーの間の関係は1カ所をのぞいて対等であるということである。唯一対等ではない点は、磁気エネルギーから熱エネルギーへの変換には、ηj^2 というオーム加熱効果があるが、その逆のルートはないという点である。これがプラズマを特徴づけている一つの大きな要素である。

シミュレーションを行う際に、これらエネルギーの時間発展も同時に解くことによって、数値誤差によるエネルギーの保存状態の良否がわかるだけでなく、物理的にも、どのような変化が系に起こっているのかがわかって、理論的解釈の助けとなる。

3.3 時間積分法

ここから時間積分法について述べるが、簡単のために、次に示す1次元の波動方程式を用いるが、最初に紹介した磁気流体方程式でも全く事情は同じである。

$$\frac{\partial u}{\partial t} = V_p \frac{\partial u}{\partial x} \tag{3.10}$$

ここで、V_p は位相速度で、初期に各格子点（ $x=x_j=\Delta x\cdot(j-1), j=1\sim N_x : \Delta x$ ； 格子間隔 ）における u が与えられる。問題は、時刻 $t=t_n(t_n=n\Delta t : \Delta t$ ； 時間間隔 ）での値から $t=t_{n+1}$ での値を求めることにある。

3.3.1 2ステップ Lax-Wendroff 法

まず初めに、2ステップ Lax-Wendroff 法を紹介する。この手法は古典的な時間積分法で、長年にわたってシミュレーション研究者に用いられてきた。この手法では、第1段階 $u_{j+1/2}$ を計算する。これは次

のように計算される。

$$u_{j+1/2}^{n+1/2} = \frac{u_{j+1}^n + u_j^n}{2} + \frac{V_p\Delta t}{2}\cdot\frac{u_{j+1}^n - u_j^n}{\Delta x} \tag{3.11}$$

第2段階として、上で求めた中間点での値を基に、本来求めたかった u_j^{n+1} を以下のように計算する。

$$u_j^{n+1} = u_j^n + V_p\Delta t\cdot\frac{u_{j+1/2}^{n+1/2} - u_{j-1/2}^{n+1/2}}{\Delta x} \tag{3.12}$$

この2つのプロセスを繰り返すことにより、時間積分を行っていく（2つの繰り返すことが2ステップと呼ばれるゆえんである)。
この手法の数値的安定性、言葉を換えると数値拡散がどの程度大きいのかを見るために、一度の積分でどの程度、波動が数値的に減衰あるいは増幅されるかを確認してみよう。最初に波が $exp(ikx)$ に比例しているとする。すると、第1段階では、

$$u_{j+1/2}^{n+1/2} = u_j^n\cdot[\{exp(ik\Delta x)+1\}/2 + \frac{\alpha}{2}\{exp(ik\Delta x)-1\}] \tag{3.13}$$

$$u_{j-1/2}^{n+1/2} = u_j^n\cdot[\{1+exp(-ik\Delta x)\}/2 + \frac{\alpha}{2}\{1-exp(-ik\Delta x)\}] \tag{3.14}$$

となる。ここで、$\alpha = V_p\Delta t/\Delta x$ である。第2段階では、

$$u_j^{n+1} = u_j^n\cdot[1 + i\alpha\sin(k\Delta x) + \alpha^2\{\cos(k\Delta x)-1\}] \tag{3.15}$$

$$= G\cdot u_j^n \tag{3.16}$$

となる。ここで G は「Gファクター」と呼ばれる数値的減衰あるいは増幅要素である。これを計算してみるとわかるが、この手法では、短い波長の波は大きく減衰する。図 3.1 の破線が、空間グリッド幅の8倍だけの距離を波が進んだ時のGファクターの α 依存性（左図）と Δx 依存性（右図）を示している。今は減衰項の存在しない波の伝播方程式を解いているのであるから、理想的にはGファクターの絶対値は常に1であるべきである。実際には、用いた手法の数値誤差のため、$\alpha > 1.0$ では1より大きく、$\alpha < 1.0$ では1より小さくなっている。数値的に安定で誤差が少ない計算のためには、Gファクターの絶対値が1より小さく、1に近いことが求められる。この数値的安定化条件（今の場合、$\alpha < 1.0$）はクーラン条件とも呼ばれている。即ち、$\alpha > 1.0$ では、数値的に不安定で時間の経過とともに波の振幅が大きくなり、いずれは発散する。一方、$\alpha < 1.0$ では数値的に安定であるが、数値誤差による波の減衰が発生する。例えば、$\alpha = 0.8$ の時、 波長 $8\Delta x$ の波動は、その波長と同じ距離を伝搬しただけで、元の振幅の 90.5% にまで落ちてしまう。これは、この手法が数値的には安定であるものの、波動の間の相互作用を見る意味では、人工的要素が非常に大きくなることを意味している。また、このために数値的粘性作用も出てきて、実際には起こらないはずの渦が現れたりする。この欠点を克服するために、新たな高精度シミュレーション手法を開発した。それを次に紹介する。

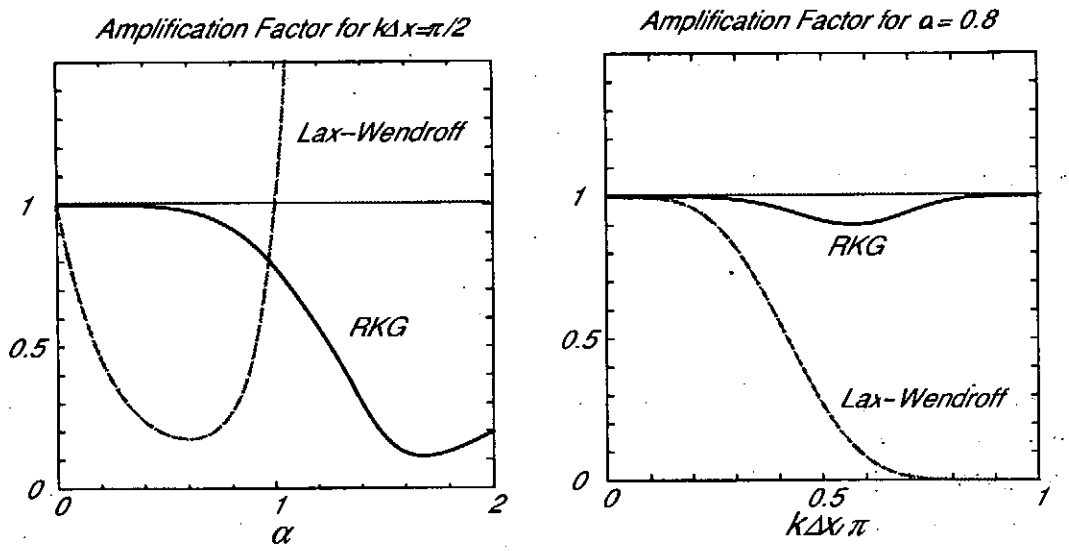


図 3.1: $k\Delta x = \pi/2$ の場合の G ファクターの α 依存性（左図）および $\alpha = 0.8$ の場合の G ファクターの Δx 依存性 (右図)。

3.3.2 高精度シミュレーション手法

この手法は、本質的には、伝統的積分法と知られている４次の Runge-Kutta-Gill 法であるが、誰も（おそらく）、これを磁気流体シミュレーションに用いようとはしなかったものである。空間微分は、通常の４次の中心差分法を用いる。この手法は、以下の４つの段階から構成される。

$$d_{j,1} = \Delta t \cdot \frac{-(u^n_{j+2} - u^n_{j-2}) + 8(u^n_{j+1} - u^n_{j-1})}{12\Delta x} \tag{3.17}$$

$$f_{j,1} = u^n_j + 0.5 \cdot d_{j,1} \tag{3.18}$$

$$q_{j,2} = d_{j,1} \tag{3.19}$$

$$d_{j,2} = \Delta t \cdot \frac{-(f_{j+2,1} - f_{j-2,1}) + 8(f_{j+1,1} - f_{j-1,1})}{12\Delta x} \tag{3.20}$$

$$f_{j,2} = f_{j,1} + (1 - \sqrt{0.5})(d_{j,2} - q_{j,2}) \tag{3.21}$$

$$q_{j,3} = 2(1 - \sqrt{0.5})d_{j,2} + 2(3\sqrt{0.5} - 2)q_{j,2} \tag{3.22}$$

$$d_{j,3} = \Delta t \cdot \frac{-(f_{j+2,2} - f_{j-2,2}) + 8(f_{j+1,2} - f_{j-1,2})}{12\Delta x} \tag{3.23}$$

$$f_{j,3} = f_{j,2} + (1 + \sqrt{0.5})(d_{j,3} - q_{j,3}) \tag{3.24}$$

$$q_{j,4} = 2(1 + \sqrt{0.5})d_{j,3} - (3\sqrt{0.5} + 2)q_{j,3} \tag{3.25}$$

$$d_{j,4} = \Delta t \cdot \frac{-(f_{j+2,3} - f_{j-2,3}) + 8(f_{j+1,3} - f_{j-1,3})}{12\Delta x} \tag{3.26}$$

$$u^{n+1}_j = f_{j,3} + d_{j,4}/6 - q_{j,4}/3 \tag{3.27}$$

この手法におけるGファクタを計算すると、

$$G = 1 + H + H^2/2 + H^3/6 + H^4/24 \tag{3.28}$$

$$H = -i\frac{\alpha}{3}[4-\cos(k\Delta x)]\sin(k\Delta x) \tag{3.29}$$

となる。図 3.1 の実線が、空間グリッド幅の8倍だけ波が進んだ時のGファクターの α 依存性（左図）と Δx 依存性（右図）を示している。前の **Lax-Wendroff** 法と同じ条件、すなわち、 $\alpha = 0.8$ の時、波長 $8\Delta x$ の波動は、その波長と同じ距離を伝搬した時、元の振幅の 99.6% を維持している。もっと波長の短い波、$4\Delta x$ の波長の波では、その波長の2倍の距離を伝搬した時、前者の手法では、たった 27% になってしまうのに対し、後者の手法では、 91.5% を維持している。我々のシミュレーショングループでは、ほとんど全てのシミュレーション研究で、この高精度手法を採用している。

3.4 終わりに

最後に、シミュレーション研究を行う上での全般的な注意点を挙げておく。1）シミュレーションの目的は、様々な要因が複雑に絡み合った自然現象の起承転結を明らかにし、その物理法則を見い出すところにある。従って、シミュレーションの結果、得られたデータそのものが自然界あるいは実験室の現実と完全に一致する必要性は必ずしもなく、得られた結果が現象を支配している物理法則の本質を捉えていることが重要である。現実とシミュレーションの見かけの結果を一致させるために、様々な人工的な数値効果が導入されたシミュレーションもあるが、これは本末転倒というべきものであろう。2）考えられる効果を全て入れるのではなく、まず理論的によく考えて、そのエッセンスを抽出したものを基礎となるシミュレーションモデルとして作り上げる。何でもかでも一緒くたに入れると、計算そのものがうまく行かないばかりでなく、現象の本質を見失ってしまう。その他の効果はあくまでも後で附加していくものであることに留意していただきたい。3）プラズマ物理では、境界条件が本質的であることが普通である。時々、簡単のためという理由だけで周期境界条件や固定境界条件を採用したシミュレーション（と称するもの）があるが、これらは往々にして虚偽とも言うべき結果を与える。いかに境界条件が難しくとも、その境界条件をコンピュータ上に作り出していくことが大切である。4）シミュレーション研究をこれから始めようという人たちにくれぐれも言っておきたいのは、シミュレーション結果が真に意味を持った結果であるか否かを判断するのは、最後は研究者の物理的直感しかないということである。従って、物理学及び数学の素養が何より大事であることを覚えておいていただきたい。

ペタフロップス・ペタバイトのコンピュータも現実のものになりつつあることを考えるとき、コンピュータシミュレーションが、これまで数学を唯一の道具として発展してきた物理学に新しい方向を与え、物理学の新展開を促進することは疑いのないところである。自然の法則に対する見方、表現法も、これまでの数式を中心とするものから大きく変革し、人の思考法にも大きな転換をもたらすと考えてよいであろう。そこに向け、より多くの研究者がこの新しい研究法に取り組んでいただくことを強く願う次第である。

3.5 演習問題

1. $x = x_j$ の周りでの2次のテイラー展開を用いて、以下の式を導け。

$$u(x_j + \epsilon) = u_j + \frac{u_{j+1} - u_{j-1}}{2\Delta x}\epsilon + \frac{u_{j+1} - 2u_j + u_{j-1}}{2\Delta x^2}\epsilon^2$$

ここで、

$$u_{j+1} = u(x_j + \Delta x),\ u_j = u(x_j),\ u_{j-1} = u(x_j - \Delta x)$$

とする。

2. 上で求めた式を用いて、半グリッド上での差分の式を導け。

$$\left(\frac{\partial u}{\partial x}\right)_{j+1/2} = \frac{u_{j+1} - u_j}{\Delta x},\ \left(\frac{\partial u}{\partial x}\right)_{j-1/2} = \frac{u_j - u_{j-1}}{\Delta x}$$

3. 時間 n の時に、$u^n(x)$ の空間プロファイルが以下の平面波で記述されているものとして、(15) 式で与えられるGファクターを求めよ。

$$u^n(x) = u_0 \exp(ikx)$$

4. $x = x_j$ の周りでの4次のテイラー展開を用いて、以下の4次精度の中央空間差分式を導け。

$$\left(\frac{\partial u}{\partial x}\right)_j = \frac{-u_{j+2} + 8u_{j+1} - 8u_{j-1} + u_{j-2}}{12\Delta x}$$

5. 時間 n の時に、$u^n(x)$ の空間プロファイルが以下の平面波で記述されているものとして、(28) 式で与えられる4次精度 RKG 法による時間積分のGファクターを求めよ。

$$u^n(x) = u_0 \exp(ikx)$$

3.6 シミュレーション実習の手順

以下の手順で、MHD シミュレーションの実習を行う。

1. cgwin
2. startx
3. ssh sekirei.theory.nifs.ac.jp
4. 作業用ディレクトリィへの移動 : cd MHD2003
5. コンパイルして実行ファイルの作成 : make
6. パラメータ値の設定・変更 : vi data01

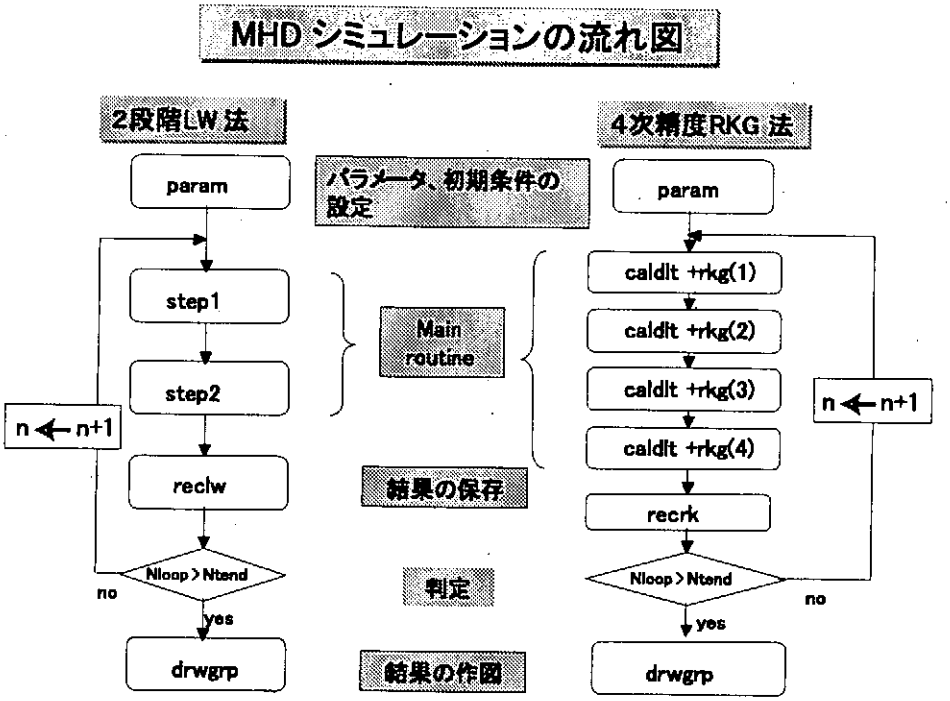
   &DATA01 Mdnmbr=16, alpha=0.80, Fname='sample', &END

   Mdnmbr : モード数(整数) ($=2\pi/(k\Delta x)$)

   alpha : クーラン数(実数) ($=V_p\Delta t/\Delta x$)

   Fname : グラフィックファイルの名前(10文字)
7. 実行 : ./mhd < data01
8. 結果のグラフィック表示 : gv sample.ps
9. 6、7、8の作業を繰り返す。

3.7 MHD シミュレーションの流れ図



第4章　プラズマ粒子シミュレーション入門

4.1　はじめに

プラズマは自由に動き回る極めて多くの数の荷電粒子の集合体であり，これらの粒子のつくる電磁場により互いに影響を与えあって運動する。プラズマ粒子シミュレーションはこの荷電粒子と電磁場の時間発展を自己無撞着に解くことによってプラズマの振る舞いを明らかにしようという手法である．しかしながらプラズマを構成する粒子の数は、実験室プラズマで $1\mathrm{cm}^3$ あたり 10^{10} 程度以上あり、すべての荷電粒子を計算機で取り扱うことは非常な困難を伴う．そこで多数の粒子をひとまとめにした超粒子というものを導入し，その超粒子とそれらのつくる電磁場の相互作用を解いていく．ただし，近接相互作用を人為的に減らすためにこの超粒子に有限の大きさを持たせ，さらに計算量を減らすため空間格子を導入して場の量をこの格子点上のみで計算するなどの手法が用いられる．本教育講座では静電粒子シミュレーションについて具体的に解説する。静電粒子シミュレーションは，荷電粒子の運動，それらのつくる静電場，外部磁場等を取り入れたプラズマ粒子シミュレーションで，イオン音波、ドリフト波などの静電波および関連した運動論的不安定性，シース，ダブルレイヤーなどの静電電位構造に関連した研究に用いられている．

なお，粒子シミュレーションでは非常に多くの粒子を取り扱う必要があるため用いる手法は精度と計算コストの兼ね合いで決める必要がある．粒子の運動方程式の差分に蛙跳法が用いられるのは，差分の精度を必要以上にあげるより，粒子の数を増して統計的な精度をあげた方がよいという考えに基づいている．同様な考えから粒子の電荷の格子点への分配も一次補間に対応する方法がよく用いられる．

最初にシミュレーションコードの構成について概観したあと，個々の粒子の座標から空間格子での電荷密度の求め方，格子点上の電荷密度から電場を求める方法，運動方程式の差分，初期値の設定方法，さらに，境界での粒子の吸収や粒子入射の手順を，一次元の静電シミュレーションについて解説する．

4.2　シミュレーションコードの構成

静電粒子シミュレーションでは，荷電粒子の運動方程式

$$m\frac{d^2\boldsymbol{x}}{dt^2} = q\left(\boldsymbol{E}(\boldsymbol{x}) + \boldsymbol{v}\times\boldsymbol{B}(\boldsymbol{x})\right) \tag{4.1}$$

と粒子のつくる電場を求める

$$\nabla^2\phi = -\frac{\rho}{\varepsilon_0} \tag{4.2}$$

$$\boldsymbol{E} = -\nabla\phi \tag{4.3}$$

を差分化して数値的に解いていく．具体的には図 4.1 のように，

1．粒子座標から格子点での電荷密度を求める．

2．ポアッソン方程式を解いて格子点上での電位を求め，さらにそれから電場を求める．

3．格子点での電場から個々の粒子に加わる力を求め，その力で粒子を加速し，移動する．

という手順を繰り返す。最初に，一次元で磁場のない場合について詳しくその手順を解説することにする。

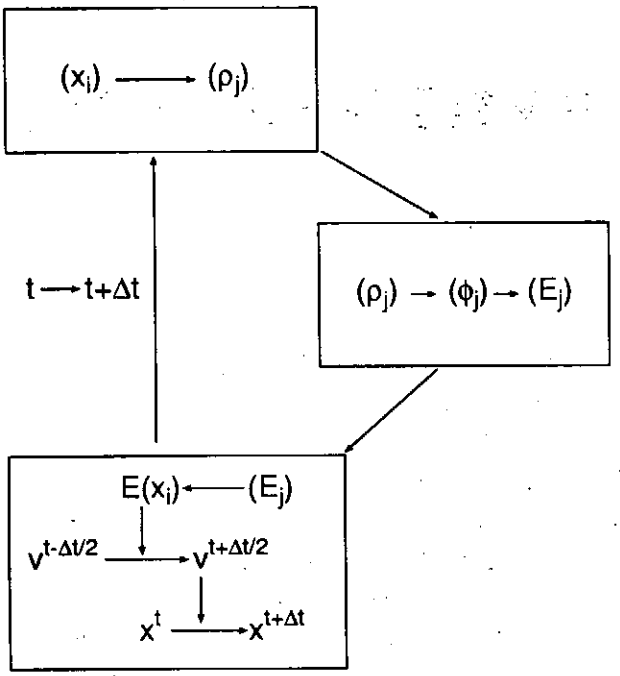


図 4.1: 静電粒子シミュレーションのアルゴリズム

4.3 格子点上での電荷密度の求め方

粒子座標から空間格子上に電荷を割り振る方法は幾つかあるが，ここでは一次補間に対応する PIC (particle in cell)[1, 2] あるいは CIC (cloud in cell)[3] と呼ばれる方法について説明する．この両者の方法の実際の手続きは全く同一で [4]，最近では PIC 法の呼び名がよく使われます．

長さ L の系を NG 等分して格子間隔は Δx ，j 番目の格子の座標を $X_j = j\Delta x$，$(j = 0, \cdots, NG)$ とし，粒子種 s 粒子（超粒子）の電荷を q_s，その i 番目の粒子の座標を $x_{s,i}$ とすると，格子点での電荷密度は，

$$\rho_j = \sum_s \frac{q_s}{\Delta x} \sum_i S(x_{s,i} - X_j) \tag{4.4}$$

となる．ただし，$S(x)$ は分配関数で，一次補間に対応して PIC 法では

$$S(x) = \begin{cases} \frac{|\Delta x - x|}{\Delta x} & |x| < \Delta x \\ 0 & |x| > \Delta x \end{cases} \tag{4.5}$$

である．実際には，粒子の座標に最も近い二つの格子点に粒子と格子点の距離に応じて電荷を割り振ることになる．なお，境界の格子 $(j = 0, j = NG)$ の電荷密度は境界条件と矛盾のないように処理する必要がある．たとえば周期境界条件の場合，電荷の分配が終了した後，$\rho_0 + \rho_{NG} \to \rho_0$ としておく必要がある．

なお，空間格子の幅 Δx は数値的不安定性を避けるために Δx □ λ_{De}(λ_{De} ： デバイ長）ととる必要がある。

4.4 格子点上での電場の求め方

格子点上で与えられた電荷密度から格子点上での電場の求め方は種々ある。一次元の場合は電荷密度から直接電場を求める方法が用いられることもあるが、多次元への拡張を考えて，ポアッソン方程式を解い

て静電ポテンシャルを求め，さらにその静電ポテンシャルから電場を求める方法を説明する。

周期境界条件の場合，フーリエ変換を用いて以下のようにして静電ポテンシャルを得ることができる。一次元の場合 ∇^2 は $\partial^2/\partial x^2$ であり，フーリエ空間では $-k^2$ となる．そこで，電荷密度分布からフーリエ変換によりフーリエ空間での電荷密度分布を求め，$\tilde{\phi}(k) = \tilde{\rho}(k)/k^2$ (k は波数) よりフーリエ空間での電位分布を求める．これを逆変換して静電ポテンシャルを得ることができる．

電荷密度は格子点上でのみ与えられているので，微分方程式の代わりに差分方程式を解くことになる．ポアッソン方程式を中心差分を用いて差分化すると，

$$\frac{\phi_{j+1} - 2\phi_j + \phi_{j-1}}{(\Delta x)^2} = -\frac{\rho_j}{\varepsilon_0} \tag{4.6}$$

となり，この差分方程式を解いて得られた格子点上での静電ポテンシャルから

$$E_j = -\frac{\phi_{j+1} - \phi_{j-1}}{2\Delta x} \tag{4.7}$$

と，格子点上の電場 E_j を求めることができる．

差分化されたポアッソン方程式を離散フーリエ変換を用いて解く．このとき，$E(0) = E(L)$，から系の内部の正味の電荷は零，すなわち $\sum_{j=0}^{NG-1} \rho_j = 0$ である必要がある．

離散フーリエ変換の数値アルゴリズムとしては高速フーリエ変換法 (FFT) (たとえば [5] を参照のこと) を利用すれば非常に高速に計算することができる．格子点上の電荷密度 ρ_j から離散フーリエ変換

$$\tilde{\rho}_n = \Delta x \sum_{j=0}^{NG-1} \rho_j e^{-ikX_j}, \quad k = \frac{2\pi n}{L} \tag{4.8}$$

により $\tilde{\rho}_n$ ($n = -NG/2, \cdots, NG/2-1$) をもとめ式 (4.6) をフーリエ変換した

$$\tilde{\phi}_n = \frac{\tilde{\rho}_n}{\varepsilon_0 K^2} \tag{4.9}$$

$$K_n^2 = k^2 \left[\frac{\sin \frac{k\Delta x}{2}}{\frac{k\Delta x}{2}} \right]^2, \quad k = \frac{2\pi n}{L} \tag{4.10}$$

により $\tilde{\phi}_n$ を求め，さらにこれを逆変換

$$\phi_j = \frac{1}{L} \sum_{n=-NG/2}^{NG/2-1} \tilde{\phi}_n e^{ikX_j}, \quad k = \frac{2\pi n}{L} \tag{4.11}$$

することによって格子点上の静電ポテンシャル ϕ_j を得ることができる．

ここで系内の全電荷が零であることから $\tilde{\rho}_0 = 0$ となることに注意されたい．また ρ_j 及び ϕ_j が実の量であることを使うと FFT での計算量を減らすことが可能である．

両端接地の境界条件の場合は，フーリエ変換の代わりに sin 変換

$$\tilde{\rho}_n = \Delta x \sum_{j=1}^{NG-1} \rho_j \sin(kX_j), \qquad k = \frac{\pi n}{L} \tag{4.12}$$

を用いてポアッソン方程式を解くことができる．この場合は系内の正味の電荷が零である必要はない．また，境界で電場が零の境界条件の場合は cos 変換

$$\tilde{\rho}_n = \Delta x \sum_{j=0}^{NG-1} \rho_j \cos(kX_j), \qquad k = \frac{\pi n}{L} \tag{4.13}$$

を用いれば同じ手順で解くことができる．ただしこの境界条件の場合，粒子の電荷を空間格子に割り振ったのち，境界での電荷密度を 2 倍にしておく必要がある．これは $-L < x < L$ の系を考えて $x = 0$ で対

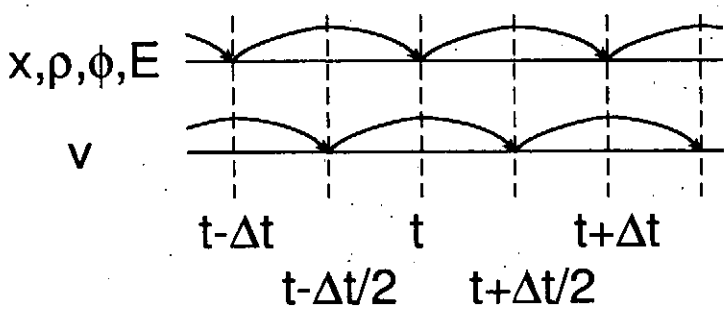


図 4.2: 蛙跳法

称と仮定していることに対応している．なお，sin 変換，cos 変換ともに FFT の手法を使って行えるので，非常に高速に計算することが可能である [5]．

ポアッソン方程式を解く際にフーリエ変換の手法を用いることの利点は，高速であることの他に，空間的なフィルターをフーリエ空間でかけることができる，電荷密度や静電ポテンシャルのフーリエ成分が同時に得られるといったことがあげられる．

4.5 運動方程式の積分

格子点上での電場が求められたら，次に運動方程式から粒子の加速，移動をする。運動方程式は二つの一階の微分方程式に分解し，蛙跳法 (leap-frog) を用いて差分化する．

$$\frac{v_{s,i}^{t+\Delta t/2} - v_{s,i}^{t-\Delta t/2}}{\Delta t} = \frac{q_s}{m_s} E(x_{s,i}^t) \tag{4.14}$$

$$\frac{x_{s,i}^{t+\Delta t} - x_{s,i}^t}{\Delta t} = v_{s,i}^{t+\Delta t/2} \tag{4.15}$$

ここで，粒子位置での電場は

$$E(x_{s,i}) = \sum_j S(x_{s,i} - X_j) E_j \tag{4.16}$$

より求め，式 (4.14) から速度を $t-\Delta t/2$ から $t+\Delta t/2$ まで進め，その速度を使って (4.15) によって粒子座標を t から $t+\Delta t$ まで進める (図 4.2)．ここで $S(x)$ は粒子の電荷の格子点への分配のときと同じようにとり，PIC 法では，粒子に最も近い二つの格子点の電場の値から線形補間により粒子座標での電場を計算する．

時間ステップ幅は数値的安定性の要請，さらには精度を確保するために，$\omega_{pe}\Delta t = 0.2$ (ω_{pe}: プラズマ周波数) 程度にとられる．

4.6 初期条件の設定

粒子シミュレーションの初期設定としては，系内をプラズマで満たした状態，粒子がない状態のどちらも可能である。初期条件として系がプラズマで満たされた状態を与える場合，$t=0$ での粒子の速度，座標を決める必要がある．与えられた粒子の速度分布 $f_0(v)$，空間分布 $n_0(x)$ から個々の粒子の速度，座標

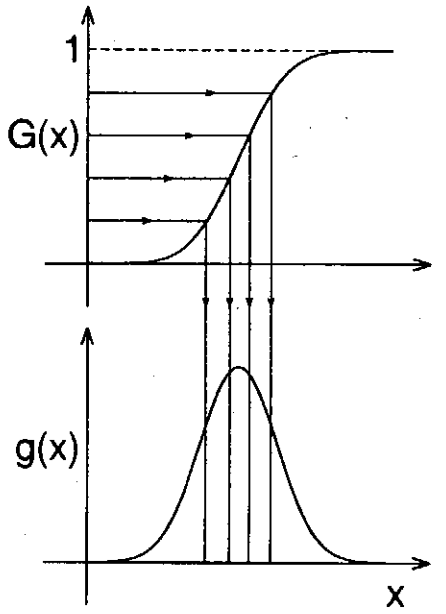


図 4.3: 累積分布関数

を決めるには累積分布関数 (cumulative distribution function) を使う方法が一般的である．任意の分布関数 $g(x)$ ($a \square x \square b$) に対して累積分布関数は

$$G(x) = \frac{\int_a^x g(x')dx'}{\int_a^b g(x')dx'} \tag{4.17}$$

と定義できる．ここで $G(a) = 0$, $G(b) = 1$ となる．N 個の粒子の分布を決めるとすると，$0 < R < 1$ の範囲で一様になるように N 個の数を順番 (N 個の一様乱数でもよい) につくり，G にあてはめ，それに対応する x を順番につくっていく．そうすれば，最初に指定した分布関数 $g(x)$ に対応する粒子の組が生成される (図 4.3)．

4.7 境界での粒子の取り扱い方

ここで粒子の境界での取り扱いについて説明する．周期境界条件では，同じ系のコピーが連なっていると考えればよいので，系の長さを L として $x = L$ の境界から粒子が飛び出したときは $x \to x - L$, $x = 0$ の境界から飛び出したときは，$x \to x + L$ とする．

次に単純な境界条件は反射条件で，$x = 0$ の境界を粒子が横切った場合は，$x \to -x$, $v \to -v$ とする．この場合ポテンシャルの境界条件としては固定境界条件あるいは電場零の境界条件が適当である．

次に固体壁や電極とプラズマが接している場合について考える．$x = 0$ の境界が電気的に浮いている電極で，その境界を横切った粒子はその電極に吸収されると仮定する．その粒子の電荷は $x = 0$ の格子点に割り振り，そこにとどまるとする．境界条件は境界で電場零の条件を採用し cos 変換を用いてポアッソン方程式を解く．その際 $x = 0$ の格子点の電荷密度を二倍しておく必要がある．これは cos 変換が $-L < x < L$ の系を仮想し，$x = 0$ で対称としていることに対応しているからである．

固定境界条件で sin 変換を用いてポテンシャルを求める場合は，境界の電荷密度の情報は用いないので，境界を横切った粒子は単に系から取り去るだけで十分である．

境界から系内に指定した分布関数で粒子を入射する手順について説明する．個々の粒子の速度は，初期設定の際に説明した累積分布関数を使って行える．分布関数 $f(v)$ に対応した速度分布で粒子を入射する場合 $F(v)=\int_0^v v'f(v')dv'/\int_0^\infty v'f(v')dv'$ を計算し，その逆関数から個々の粒子の速度を求める．たとえば，$f(v)$ が半マックスウェル分布の場合は，粒子の熱速度を v_t，R を $0<R<1$ の一様乱数として $v=v_t\sqrt{-2\ln R}$ から粒子の速度を求め，境界から粒子を入射する．一方，$f(v)$ がシフトマックスウェル分布などのように $F(v)$ が解析的に表わされない場合は数値的に積分して $F(v)$ を求めておき，$0<R<1$ の一様乱数 R から $F(v)$ に対応する v を求める．時間的に一定の数の粒子を入射する場合，各時間ステップ毎に入射する粒子の数はフラックス $\Gamma=\int_0^\infty vf(v)dv$ から求めておく。また，$x=0$ から入射したときの最初の座標は新たな一様乱数 R を生成し $x=v\Delta tR$ とし，入射した粒子が適度に分散するようにする．

境界が固体壁や電極ではなく無限プラズマと接している場合を模擬する場合はその境界にシースができたり擾乱が生じるのは好ましくない．このような擾乱を持ち込まずかつシースも生じない自己無撞着開境界モデルが最近開発されている [6，7]．ここでは境界を横切る粒子の正味のフラックスが時間的に一定となるように入射する粒子の数を決める．仮に，粒子の速度分布がマックスウェル分布からずれており上流から下流に流れている場合を考える．擾乱のない場合に，その速度分布で時間ステップあたりに境界を横切る正味の粒子の数を計算しておき N_{net} とする．各時間ステップ毎に上流の境界から系外に出る粒子の数 N_{out}^{up} を数え，上流の境界から入射する粒子数 N_{in}^{up} を

$$N_{in}^{up}=N_{net}-N_{out}^{up} \tag{4.18}$$

として求め，指定した速度分布関数で粒子を入射する．下流の境界に関しても同様に下流の境界から系外に出る粒子の数 N_{out}^{down} をかぞえ，下流の境界から入射する粒子数 N_{in}^{down} を

$$N_{in}^{down}=N_{out}^{down}-N_{net} \tag{4.19}$$

として求め，指定した分布関数で粒子を入射する．このとき，各時間ステップに系から出る粒子の数は系の内部の状態に依存し，時間ステップ毎に異なる．その数に応じて入射する粒子の数を調整するので系の内部の粒子の数は常に一定値となる．この方法によると境界にシースをつくらず，非物理的な擾乱を与えること無く指定した分布関数で系の内部に粒子を入射できる．このモデルでは，静電ポテンシャルに対しては電場が境界で零となる境界条件が用いらる．

4.8 おわりに

静電粒子シミュレーションについて説明してきた。陽解法による PIC 静電粒子シミュレーションは手法としてはほぼ確立されており，その性質，数値的特性は詳細に調べられている．ただし，シミュレーションコードを始めて作成した場合は，そのコードの正しさを確認するために多くのテストをする必要がある．周期境界条件の場合，エネルギー保存，揺動レベル，ランダウ減衰等も含めた波動の分散などが最初にテストする項目となる．境界から粒子を入射したり，粒子吸収壁を設定したりした場合はシースの深さ等が理論値と一致するかを確かめたりすることもある。一方，他で開発されたシミュレーションコードを用いる場合においても，その適用条件に十分な注意を払う必要がある。シミュレーションコードは適用条件，適用限界を正しく把握した上で利用して始めて物理的に正しい結果を与えるからである．

関連図書

[1] F. H. Harlow, Meth. Comput. Phys., **3**, 319, B. Akderm, S. Fernbach, and M. Rotenberg, eds., Academic, New York, 1964.

[2] R. L. Morse and C. W. Nielson, Phys. Fluid, **12**, 2418 (1969).

[3] C. K. Birdsall and D. Fuss, J. Comput. Phys., **3**, 494 (1969).

[4] C. K. Birdsall and A. B. Langdon,
Plasma Physics via Computer Simulation, McGraw-Hill Book Company, New York, 1985 and Adam Hilger, Bristol, Philadelphia and New York, 1991.

[5] W. H. Press,
S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran*, Cambridge University Press, 1986, 1992.

[6] H. Takamaru, T. Sato, R. Horiuchi, K. Watanabe, and the Complexity Simulation Group, J. Phys. Soc. Jpn., **66**, 3826 (1997).

[7] S. Ishiguro, T. Sato, H. Takamaru, and the Complexity Simulation Group, Phys. Rev. Lett., **78**, 4761 (1997).

Recent Issues of NIFS-PROC Series

NIFS-PROC-35 T. Watari,
Plasma Heating and Current Drive: Oct. 1997

NIFS-PROC-36 T. Miyamoto and K. Takasugi (Eds.)
Production and Physics of High Energy Density Plasma: Production and Physics of High Energy Density Plasma: Oct. 1997

NIFS-PROC-37 (Eds.)T. Fujimoto, P. Beiersdorfer,
Proceedings of the Japan-US Workshop on Plasma Polarization Spectroscopy and The International Seminar on Plasma Polarization Spectroscopy January 26-28, 1998, Kyoto: June 1998

NIFS-PROC-38 (Eds.) Y. Tomita, Y. Nakamura and T. Hayashi,
Proceedings of the Second Asian Pacific Plasma Theory Conference APPTC '97, January 26-28, 1998, Kyoto: Aug. 1998

NIFS-PROC-39 (Ed.) K. Hirano,
Production, Diagnostics and Application of High Energy Density Plasmas: Dec. 1998

NIFS-PROC-40 研究代表者　加古　孝（電気通信大学）
所内世話人　渡辺　二太
平成 10 年度核融合科学研究所共同研究 研究会「プラズマ閉じ込めに関連する数値計算手法の研究」
Ed. by T. Kako and T. Watanabe
Proceeding of 1998-Workshop on MHD Computations "Study on Numerical Methods Related to Plasma Confinement Apr. 1999

NIFS-PROC-41 (Eds.) S. Coto and S. Yoshimura,
Proceedings of The US-Japan Workshop and The Satellite Meeting of ITC-9 on Physics of High Beta Plasma Confinement in Innovative Fusion System, Dec. 14-15, 1998, NIFS, Toki: Apr. 1999

NIFS-PROC-42 (Eds.) H. Akiyama and S. Katsuki,
Physics and Applications of High Temperature and Dense Plasmas Produced by Pulsed Power: Aug. 1999

NIFS-PROC-43 (Ed.) M. Tanaka,
Structure Formation and Function of Gaseous, Biological and Strongly Coupled Plasmas: Sep. 1999

NIFS-PROC-44 (Ed.) T. Kato and I. Murakami,
Proceedings of the International Seminar on Atomic Processes in Plasmas, July 29-30, 1999, Toki, Japan: Jan. 2000

NIFS-PROC-45 (Eds.) K. Yatsui and W. Jiang,
Physics and Applications of Extreme Energy-Density State, Nov. 25-26, 1999, NIFS: Mar. 2000

NIFS-PROC-46 研究代表者　加古　孝（電気通信大学）
所内世話人　渡辺　二太
平成 11年度核融合科学研究所共同研究 研究会「プラズマ閉じ込めに関連する数値計算手法の研究」
Ed. by T. Kako and T. Watanabe
Proceeding of 1999-Workshop on MHD Computations "Study on Numerical Methods Related to Plasma Confinement June. 2000

NIFS-PROC-47 岡本正雄、村上定義、中島徳嘉、汪衛生
プラズマ物理におけるモンテカルロシミュレーション
WatanabeM. Okamoto, S. Murakami, N. Nakajima, W.X. Wang,
Monte Carlo Simulations for Plasma Physics: July 2000

NIFS-PROC-48 K. Miyamoto,
Fundamentals of Plasma Physics and Controlled Fusion: Oct. 2000

NIFS-PROC-49 (Ed.) K. Kawahata,
Proceeding of the 5th International Workshop on Reflectometry, 5-7 march, 2001: May 2001

NIFS-PROC-50 (Ed.) S. Ishii
Workshop on Extremely High Energy Density Plasmas and Their Diagnostics, Mar. 8-9, 2001, National Institute for Fusion Science, Toki, Japan: Sep. 2001

NIFS-PROC-51 (Ed.) K. Horioka,
Physics and Applications of High Energy Density Plasmas -Extreme state driven by pulsed electromagnetic energy, Dec.20-21, 2001, National Institute for Fusion Science; June 2002

NIFS-PROC-52 第 6 回「シミュレーション・サイエンス・シンポジウム」及び核融合科学研究所共同研究「大型シミュレーション研究」合同研究会　集録
Proceedings of Joint Meeting of the 6th Simulation Science Symposium and the NIFS Collaboration Research "Large Scale Computer Simulation": Mar. 2003

NIFS-PROC-53 研究代表者　渡邉二太
LHD型磁場配位を用いたICRF支援　水素・硼素核融合炉の理論的研究ー平成14年度核融合科学研究所共同研究ー
(Ed.) T. Watanabe
Theoretical Study for ICRF Sustained LHD Type p-^{11}B Reactor: Apr. 2003

NIFS-PROC-54 (Ed.) K. Masugata
Physics and Applications of Micro and Fast Z-Pinch Plasmas (Dec. 5-6, 2002, NIFS): July 2003

NIFS-PROC-55 シミュレーション科学教育講座 2003 テキスト
Text of Simulation Science Open Lecture 2003: Apr. 2004