

計算物理学実習

2021 年度物理学科 秋学期

担当 山内 淳

0 はじめに (高野 宏)

0.1 背景

慶應義塾大学理工学部物理学科では1995年度より、計算物理学の講義、実習を始めました。この科目を開設した背景は次のようなものです。

1. 計算機の進歩(高速化、低価格化、ネットワーク化)にともない、物理学において計算機を使う機会が多くなってきたこと。
例えば、数値計算、数式処理、実験データの処理、実験装置の制御、グラフ処理、データの可視化、論文清書などの文書処理、文献検索などのデータベースの利用、電子メールなどネットワークによる情報交換など。
2. 新しい物理学の分野としての計算物理学の成立。
これまで解析的理論や実験で扱うことのできなかった現象を、計算機を使うことによって、初めて扱うことができるようになってきた。
3. ワークステーションが重要性を増してきている。
ワークステーションと呼ばれる計算機は、計算能力、ネットワーク能力、グラフィック能力が高く、コストパフォーマンスが良いため、近年、物理学の分野でも利用する機会が多くなってきている。
4. 学部専門教育において、物理学における計算機利用に関する教育および利用できる計算機の整備が必要になってきている。
実際、学部4年での卒業研究等において計算機を利用する機会が多く、計算機利用に対する各学生の知識、経験を増す必要がある。高性能の計算機を各学生が自由に使えるようにし、計算機を用いるような課題に全員がとりくむことができるようにする必要がある。

0.2 目標

この科目では、次のことを目標としています。

- 計算機(ワークステーション)の基本的操作方法を習得する。
- プログラミング言語、プログラムの書き方など、プログラミングの基本を習得する。
- 計算物理学の基本的方法を習得する。
- 例題、実習を通して物理の理解を深める。

このほか、グラフィック処理、ネットワーク利用についても学んでいきます。

0.3 心構え

計算機を使う本来の目的は、計算機を道具として(ここでは物理学に)役立てるということです。どのような道具を使っているか(どのような計算機を使っているか、どのようなソフトウェアを使っているか、どのようなプログラミング言語をつかっているかというようなこと)が重要なのではなく、その道具をいかに使いこなして役立てることができたかが重要なことです。

この科目では、計算物理学の基本的方法の習得のための基礎として、現在物理学の研究環境で一般的と思われる計算機やプログラミング言語などの使い方を学びます。しかし、ここで習った計算機やプログラミング言語が今後ずっと物理学の研究環境で一般的であり続けるとは限りません。また、今後各人が利用できる計算機の環境も、この科目で用いた環境とは異なっているでしょう。急速に進歩し変化していく計算機という道具に柔軟に対応し、利用できる計算機環境に柔軟に適応して、計算機を使う本来の目的を達するように心がけましょう。

1 ワークステーションの基本的操作法 (高野 宏)

ここでは、ワークステーションと呼ばれる計算機の基本的な使い方について紹介します。

1.1 ワークステーションの構成

ワークステーションは大抵、

- 計算機本体、
計算処理を行なう部分、半導体を用いてデータを保持するメモリー、ハードディスクと呼ばれるデータを記録保存する磁気記録装置などからなっています。
- 文字や図形などの計算機の出力を画面に表示するディスプレイと呼ばれるテレビのような装置、
- 利用者が計算機に対して命令を与えるためのキーボードと呼ばれるタイプライターのような装置、
- 計算機に対してディスプレイ画面上での位置を指定したり、命令を与えたりするためのマウスと呼ばれる装置

などから成っています。

通常、ワークステーション本体の電源は入れたままにしておきます。ワークステーションの管理をしている人以外の一般の利用者がワークステーション本体の電源を入れたり切ったりすることはありません。

1.2 ワークステーションの OS

計算機はどのように動作するかを記述したプログラムに従って動きます。計算機の利用者は普通、オペレーティング・システム (略称 OS) と呼ばれるプログラムを介して計算機を操作します。利用者からは、OS は計算機を操作するための約束ごとを決めたプログラムと見ることができます。ワークステーションでは UNIX と呼ばれる OS を採用したものが現在は主流になっています。UNIX は次のような特徴をもっています。

- OS として UNIX を採用している計算機の種類が多い。
パーソナルコンピュータからワークステーション、大型コンピュータ、スーパーコンピュータに至るまで、幅広い種類の計算機で利用することができます。

- マルチユーザの OS である。
1 台の計算機を複数の利用者が同時に使うことができます。
- マルチタスクの OS である。
1 台の計算機が複数のタスク (仕事) を同時に処理していくことができます。

1.3 UNIX の使い方

以下では、ワークステーションの基本的使い方として、UNIX の使い方を紹介します。

1.3.1 使用開始と終了

a. 利用者登録

UNIX では利用者として登録した人だけが計算機を使うことができます。登録された利用者には、利用者を区別する UNIX 上での名前 (ログインネーム、ログイン名) が与えられます。また、そのログイン名を使った人が本人であることを確認するために、本人だけが知っている合言葉 (パスワード) を登録します。

b. 使用開始


計算機を使い始めるには、ログイン (login) という手続きをします。これは、これから誰が計算機を使うかを計算機に認識させるための手続きです。

使える状態のワークステーションのディスプレイの画面には

```
login: █
```

と表示されています。この login: はログイン・プロンプトと呼ばれ、計算機がログインを受け付ける状態であることを示しています。この login: の後の █ は、画面上で点滅している四角 (または下線) を表しています。これは、カーソルと呼ばれ、次に文字を入力する場所を表しています。これに対して、次のようにログイン名とパスワードをキーボードからタイプして入力します。

```
login: b000000↵  
Password: XXXXXXXX↵
```

ここで、キーボードから入力する部分は下線をつけてあります。また、 はリターンキーを表します。パスワードの入力で XXXXXXXX となっている部分は実際にはディスプレイの画面には表示されません。ログインに成功するといくつかのメッセージが表示された後、

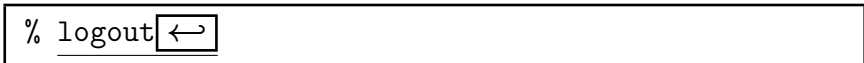

 % █

のように表示されます。この % はプロンプトと呼ばれ、計算機が命令 (コマンド) の入力を待っている状態であることを表しています。

c. 使用終了

計算機を使用を終えるには、ログアウト (logout) という手続きをします。これは、現在の利用者が計算機を使い終ったことを計算機に認識させるためのものです。

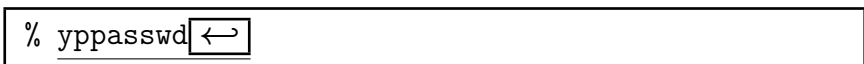

上記のプロンプトに対して、

 % logout



とタイプします。ログアウトがうまくいくと、再びログインプロンプトが表示され、次の人が計算機を使える状態になります。

d. パスワードの変更

パスワードは決して他人に知られてはいけません。理工学 ITC および物理学科のワークステーションでパスワードを変更するには、プロンプトに対して、





 % yppasswd

とタイプし、その後計算機からの指示に従って、

現在のパスワード を 1 回、
新しいパスワード を 2 回

入力します。

e. 文字の入力について

UNIX は をタイプするごとに、入力を受け付けます。 を押す前ならば、誤って入力した文字は、 (あるいは) と書かれたキーを押すことにより 1 文字ずつ取り消すことができます。

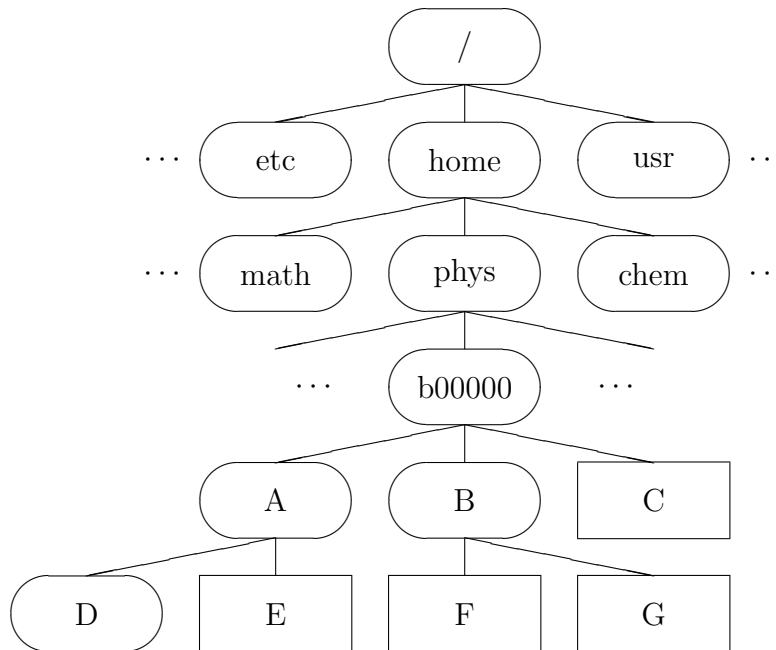


図 1.

1.3.2 ファイル・システム

a. ファイル

いろいろなプログラムやデータはワークステーションのハードディスクと呼ばれる磁気記録装置に記録し保存することができます。そのとき、ファイルという単位で名前をつけて管理します。

b. ディレクトリ

UNIX のファイルはツリー状 (枝分かれした状態) になって管理されています。例えば図 1 のようになっています。図 1 で、長円で表された枝分かれの部分はディレクトリと呼ばれる特別なファイルがあり、枝分かれした先にどのようなファイルがあるかを管理しています。ディレクトリの下にディレクトリを作ることができるのでツリー状になります。長方形で表された枝の末端の部分は通常のファイルを表しています。一番上にあるディレクトリ / はルート・ディレクトリと呼ばれます。ファイル E は、ルート・ディレクトリからたどって /home/phys/b00000/A/E という名前で指定することができます。最初の / はルート・ディレクトリを表し、2 番目以降の / は“の下の”という意味のディレクトリとディレクトリの区切り、または、ディレクトリとファイルの区切りを表します。このようにルート・ディレクトリからたどってファイルやディレクトリを指定する名前を絶対パス名といいます。ディレクトリ B は /home/phys/b00000/B という絶対パス名になります。

c. 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) といいます。カレント・ディレクトリからの相対的關係でファイルやディレクトリを指定することができます。例えば、図1でカレント・ディレクトリが A (/home/phys/b00000/A) の場合、ファイル E は ./E で指定され、ディレクトリ D は ./D で指定されます。ここで、“.” はカレント・ディレクトリを表します。このカレント・ディレクトリの下という意味の ./ は省略可能です。つまり、上の例では、./E の代わりに E を ./D の代わりに D を用いることができます。カレント・ディレクトリが D (/home/phys/b00000/A/D) の場合、ファイル F は ../../B/F で指定されます。ここで、最初の “..” はカレント・ディレクトリの一つ上のディレクトリ (/home/phys/b00000/A) を表します。(ディレクトリの上のディレクトリはただ一つしかないことに注意してください。) 次の “..” は、直前に指定されているディレクトリ (/home/phys/b00000/A) の一つ上のディレクトリ (/home/phys/b00000) を表します。このようにカレント・ディレクトリからたどってファイルやディレクトリを指定する名前を相対パス名といいます。

d. ホーム・ディレクトリ

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ます。ホーム・ディレクトリは利用者ごとに /home/phys/ログイン名のように設定されています。自分のホーム・ディレクトリを指定するのに、“~” を使うことができます。図1の例で、/home/phys/b00000 ログイン名 b00000 の人のホーム・ディレクトリであるとしします。ログイン名 b00000 の人は、ファイル F を ~/B/F と指定することができます。自分以外の人のホーム・ディレクトリを指定するのに、“~ログイン名” を使うことができます。ログイン名 b00000 の以外の人、ファイル F を ~b00000/B/F と指定することができます。

e. ディレクトリに対する操作

- カレント・ディレクトリの表示
プロンプトに対して

```
% pwd ↵
```

とタイプします。

- カレント・ディレクトリの変更

```
% cd ディレクトリ名 ↵
```


とタイプします。これによって、ディレクトリ名 (相対パス名または絶対パス名) で指定されたディレクトリがカレント・ディレクトリになります。cd とディレクトリ名の間にスペースがあることに注意してください。

```
% cd [←]
```

のようにディレクトリ名を省略した場合、ホーム・ディレクトリがカレント・ディレクトリになります。cd は **change directory** の略です。

- ディレクトリの作成

```
% mkdir ディレクトリ名 [←]
```

とタイプします。例えば、mkdir abc [←] とした場合、ディレクトリ名 abc は ./abc の意味で (相対パスを用いて、./ を省略した場合に相当)、カレント・ディレクトリの下に abc というディレクトリが作られます。mkdir は **make directory** の略です。

- ディレクトリの削除

```
% rmdir ディレクトリ名 [←]
```

とタイプします。ただし、その下に何も無い空のディレクトリでなければ削除できません。rmdir は **remove directory** の略です。

- ディレクトリのリスト

ディレクトリの下に、どのようなファイルやディレクトリがあるかを知るには、

```
% ls [オプション] [ディレクトリ名] [←]
```

または

```
% ls [オプション] [ファイル名] [←]
```

とタイプします。ここで、[] で囲んだ部分は省略可能です。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・ディレクトリの下にあるファイルやディレクトリを表示します。オプションには次のようなものがあります。

- l ファイルに関する詳細な情報 (サイズ、日付など) を出力します。
- a ディレクトリの情報を表示する場合に、通常表示されない “.” から始まる名前のファイルも表示します。
- F ディレクトリの後には / 、実行可能ファイルの後には * をつけるなど、ファイルの種類もわかるように表示します。

ls は list の略です。

f. ファイルに対する操作

- ファイルの作成

エディタ、コンパイラ (後述) などのプログラムを使って作ります。プログラムを実行した結果をファイルに出力することもあります。

- ファイルの削除

```
% rm ファイル名 ↵
```

とタイプします。

```
% rm -i ファイル名 ↵
```

とすると、指定されたファイルを本当に削除していいかどうかを尋ねてきますから、削除していい場合には `y` を入力します。普通、上のようにタイプしても下の意味になるように設定してあります。rm は **remove** の略です。

- ファイルの内容の表示

```
% cat ファイル名 ↵
```

とタイプします。複数のファイル名を指定するとそれらを連結して出力します。cat は **concatenate** の略です。

ファイルが大きい場合、cat ではディスプレイの 1 画面分に収まりきらないことがあります。その場合に便利なのが、more コマンドです。

```
% more ファイル名 ↵
```

とタイプするとファイルの内容を出力します。ディスプレイの 1 画面分を出力すると、一旦出力を停止します。そこで次のキーを押すと、以下のような動作をします。

`space` 次の 1 画面分を出力します。

`↵` 次の 1 行を出力します。

`q` 出力をそこで打ち切り more を終了します。

`h` キーの使い方の説明が表示されます。

同様のコマンドに less があります。less の場合、`k` を押すことによって、1 行分前に戻すことができます。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、標準出力に出力します。そこで、あるコマンドの標準出力への出力が 1 画面分を越えるような場合、次のようにすると便利です。

```
% コマンド | less↵
```

- ファイルの複写と移動

```
% cp 複写元のファイル名 複写先のファイル名↵
```

```
% mv 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 元のファイル名 移動先のディレクトリ名↵
```

とタイプします。後者の場合、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ます。ファイルの移動はファイル名の変更と同じことになります。cp は **copy** の略です。mv は **move** の略です。

g. ファイル名の指定

UNIX には、ワイルド・カードという文字があり、ファイル名のパターンの指定に使うことができます。それは、次のような文字です。

- * 任意の文字列を表します。
- ? 任意の 1 文字を表します。
- [] [] 内の任意の 1 文字。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 文字を表します。

これらのワイルド・カードを用いてファイル名のパターンを指定した場合、そのパターンと実際に存在するファイル名とが一致したものが指定されたことになります。

ワイルド・カードとは別に、便利なファイル名の指定方法があります。

{ } { } 内の “,” で区切られた文字列のいずれかを表します。

この場合は、パターンを表すのではなく、ファイルが実際にあるかないかにかかわらず、区切られた文字列をすべて展開したものが指定されます。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、

`t*` により `sa` 以外全てのファイルが指定され、
`t?` により `t1`、`t2`、`t3` が指定され、
`*1` により `t1`、`t11` が指定され、
`?1` により `t1` が指定され、
`t[13]` により `t1`、`t3` が指定され、
`{t1,sa}` により `t1`、`sa` が指定されます。

`{ }` はファイル名に合うパターンがない時に使うと便利です。例えば、`{A,B}/{a,c,f,p}` は `A/a.c`、`A/f.p`、`B/a.c`、`B/f.p` に展開されます。

h. ファイル関係のよく使うコマンドのまとめ

コマンド名	意味
<code>cat</code>	ファイル内容の出力 (concatenate)
<code>more</code>	ファイル内容の 1 画面ずつの出力 (more)
<code>cd</code>	ディレクトリの変更 (change directory)
<code>cp</code>	ファイルの複写 (copy)
<code>ls</code>	ファイル、ディレクトリに関する情報出力 (list)
<code>mkdir</code>	ディレクトリを作る (make directory)
<code>mv</code>	ファイルの移動 (move)
<code>rm</code>	ファイルの消去 (remove)
<code>rmdir</code>	ディレクトリの消去 (remove directory)
<code>pwd</code>	カレント・ディレクトリ名の出力 (print working directory)

1.3.3 入出力

a. 標準入力、標準出力

UNIX 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ります。
 コマンドは通常、標準入力から入力を受け、標準出力に結果を出力します。
 また、FORTRAN(f90) のプログラムで、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ます。

b. 入出力の切り換え (リダイレクション)

通常、標準入力はキーボード、標準出力と標準エラー出力はディスプレイの画面が割り当てられています。次のようにして切り換えることができます。

> ファイル名	標準出力をファイルへ (ファイルの頭から) 書き込みます。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ます。
>! ファイル名	標準出力をファイルへ (ファイルの頭から) 書き込みます。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みます。
>> 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します。
>& ファイル名	標準出力と標準エラー出力をファイルへ (ファイルの頭から) 書き込みます。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ます。
>&! ファイル名	標準出力と標準エラー出力をファイルへ (ファイルの頭から) 書き込みます。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みます。
>>& ファイル名	標準出力と標準エラー出力を既に存在するファイルの末尾へ追加します。
< ファイル名	標準入力をそのファイルから受け取ります。
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します。

これによって、キーボードより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記録したりすることが簡単にできます。

例えば、

```
% cat ファイル1 > ファイル2 
```

とすると、ファイル2にはファイル1の内容が入ります。さらに、

```
% cat ファイル1 >> ファイル2 
```

とすると、ファイル2にはファイル1の内容が追加されます。

cat コマンドはファイル名を指定しないと、標準入力から入力を受け取ります。

```
% cat 
a 
a
b 
b
ab 
ab
 CTRL-d
```

のように、キーボードからの入力 (標準入力) を 1 行ずつ受取り画面 (標準出力) に表示します。最後に入力した **CTRL-d** は、**CTRL** キーと **d** のキーを同時に押すことを意味しています。**CTRL-d** は入力の終りを表しています。

```
% cat > ファイル1↵
a↵
b↵
ab↵
CTRL-d
% cat ファイル1↵
a
b
ab
```

とすると、ファイル1にはキーボードから入力した内容が入ります。

```
UUUUUUread(*,*)i
UUUUUUwrite(*,*)i*2
UUUUUUstop
UUUUUUend
```

という FORTRAN(f90) のプログラムをファイル test1.f に入れて f90 test1.f↵ でコンパイルして作った実行可能ファイルを a.out とするとき、

```
% a.out | a.out | a.out | a.out↵
```

とタイプし、

```
1↵
```

と入力すると、

```
16
```

が出力されます。

1.3.4 マニュアル

UNIX には、コマンドの使い方等の説明 (マニュアル) をディスプレイの画面に表示する仕組みがあり、オンライン・マニュアルと呼ばれています。

```
% man コマンド名↵
```

とすることにより、そのコマンドに関する説明 (マニュアル) が表示されます。マニュアルの出力に対して more (less) コマンドが働いていますから、長いマニュアルでも 1 画面ずつみることができます。例えば、

```
% man man↵
```

とすると、man コマンド自身のマニュアルを見ることができます。

```
% man -k キーワード↵
```

とすると、キーワードに関連したコマンド名が表示されます。何かわからないことがあった場合、人に質問する前に、自分でオンライン・マニュアルを調べるようにしましょう。

1.3.5 プリンタ

自分で書いたプログラムや計算結果等、通常の文字で書かれたファイルの内容をポストスクリプト・プリンタを使って紙に印刷するには、

```
% a2ps ファイル名 | lpr -P プリンタ名↵
```

とします。a2ps は通常の文字で書かれたファイルをポストスクリプト形式のファイルに変換するコマンドです。例えば、理工学 ITC のワークステーションでは、

```
% a2ps ファイル名 | lpr -Pmono_yc↵
```

とします (-Pmono_yc は省略可能です)。プリンタは複数の利用者で共有していますから、順番に印刷していきます。指定したプリンタの印刷出力の順番を見るには

```
% lpq -P プリンタ名↵
```

とします。印刷待ち、印刷中の自分の印刷出力を取り消すには、

```
% lprm -P プリンタ名 ジョブ番号↵
```

とします。ジョブ番号は lpq で見て確認します。

```
% lprm -P プリンタ名 -↵
```

とすると、印刷待ち、印刷中の自分の印刷出力がすべて取り消されます。

2 プログラミング (高野 宏)

ここでは、FORTRAN によるプログラミングの方法を紹介します。

FORTRAN プログラムを作って走らせるためには、普通、次のような手続きが必要です。

- (1) エディタを使って、FORTRAN のプログラムの書かれたファイル (ソースファイル) を作ります。
- (2) FORTRAN コンパイラを使って、ソースファイルを機械語 (計算機の言葉) に翻訳 (コンパイル) します。この結果、機械語に翻訳されたファイル (オブジェクトファイル) が作られます。
- (3) リンカを使って、コンパイラの作ったオブジェクトファイルや、すでに用意されているオブジェクトファイルの一つに結合 (リンク) して、実行可能ファイル (計算機が直接実行できるファイル) を作ります。
- (4) 実行可能ファイルを実行します。

以下、これらの手続きに必要な事項を説明していきます。

2.1 エディタの使い方


エディタは、プログラムやデータなどをファイルとしてハードディスクに書き込んだり、すでにハードディスク上にあるファイルの内容を変更、編集したりするためのプログラムです。ここでは、UNIX の標準エディタである vi と、ワークステーション上で普及している emacs について簡単に紹介します。

理工学 ITC のワークステーションでは、普通 emacs が用いられています。しかし、計算機によっては、emacs が無く、vi のみしか使えない場合もあります。また、計算機の使われ方によっては、多くのメモリーを使用する emacs より、比較的メモリの使用量の少ない vi の方が適している場合もあります。実際、この科目の実習で、数値計算に用いる物理学科の共用の計算機 (physhprx.cm.phys.keio.ac.jp) 上では、vi しか使えません。さらに、UNIX の管理を行なうためには、vi は必ず必要となります。従って、vi は必ず使えるように練習しましょう。

2.1.1 vi

以下では vi の使い方を簡単に紹介します。vi はここで紹介した以外にも多くの機能を持っています。さらに詳しいことを知りたい場合は、詳しくは、UNIX の解説書 (坂本文, “たのしい UNIX — UNIX への招待 —” (1990, アスキー) など) や vi のマニュアルを参照してください。

- vi の起動
vi を起動するには、

% vi ファイル名 


とタイプします。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す。

- 画面
編集するファイルの内容が画面に表示されます。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ます。何も入っていない空の行には、行の左端に “~” が表示されます。新しいファイルを作ろうとしたときには、すべての行が空であるように表示されます。
- カーソル
ファイルの内容が画面に表示された時に、画面の左上隅の一文字が点滅する四角の中に表示されていますこの四角はカーソルと呼ばれ、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ます。
- カーソルの移動
次のキーをタイプすることにより、文字の書き込まれた範囲で、カーソルの位置を上下左右に動かすことができます。

<u>h</u>	← 左にカーソルを移動します。
<u>j</u>	↓ 下にカーソルを移動します。
<u>k</u>	↑ 上にカーソルを移動します。
<u>l</u> または <u>space</u>	→ 右にカーソルを移動します。

- 文字を書き込む
次のようにタイプすることにより、画面に文字を書き込むことができます。

i 書き込む文字列 <u>ESC</u>	カーソルの左側に文字列を挿入 (insert) します。
a 書き込む文字列 <u>ESC</u>	カーソルの右側に文字列を追加 (append) します。
o 書き込む文字列 <u>ESC</u>	カーソルのある行の下に新しい行を作り、文字列を書き込みます。
O 書き込む文字列 <u>ESC</u>	カーソルのある行の上に新しい行を作り、文字列を書き込みます。

ここで、ESC は ESC と書かれたキー (エスケープキー) を表します。
書き込む文字列の中で  をタイプするとそこで改行され、現在書き込ん

でいる行の下に新しい行ができ、そこに文字列が続けて書き込まれていきます。

誤って入力した文字は、**ESC** をタイプする前 (文字列を書き込んでいる状態) ならば **BS** キーを押すことにより 1 文字ずつ取り消すことができます。ただし、**↵** は取り消せません。

- 書き込まれた文字の削除

すでに書き込まれた文字を削除するには、次のようにします。

x	カーソルの位置の文字を削除します。
dd	カーソルのある行全体を削除します。

- 直前の操作の取り消し、繰り返し

直前に行なった文字の書き込みや削除の操作を、取り消したり、繰り返したりするには次のようにします。

u	直前に行なった文字の書き込みや削除の操作を、取り消す (undo)。
.	直前に行なった文字の書き込みや削除の操作を、繰り返す。

- ファイルへの書き込み、ファイルの読み込み

次のようにすることによって、編集結果を指定したファイルに書き込んだり、指定したファイルの内容を編集画面に読み込んだりすることができます。

:w ファイル名 ↵	編集結果を指定されたファイルに書き込みます。ファイル名を省略すると、現在編集画面のファイルに書き込みます。
:r ファイル名 ↵	指定されたファイルの内容をカーソルのある行の下に読み込みます。

- vi の終了

vi を終了するには次のようにします。

ZZ	編集結果をファイルに書き込み、vi を終了します。
:q! ↵	編集結果をファイルに書き込まないで、vi を終了します。

- vi のモード

vi の文字列を書き込んでいる状態をテキスト入力モードと呼び、それ以外の状態をコマンドモードと呼びます。これまで紹介した操作をまとめると下図のようになります。

UNIX のプロンプト % の出ている状態	
vi ファイル名	↓ ↑ ZZ または :q!
vi のコマンドモード	
h, j, k, l, space :カーソル移動	
x, dd :1 文字、1 行削除	
u, . :直前の操作の取り消し、繰り返し	
:w ファイル名	:r ファイル名 :ファイルの書き込み、読み込み
i, a, o, O	↑ ESC
vi のテキスト入力モード	
タイプした文字が画面に書き込まれます。	
BS :入力した文字の取り消し	

2.1.2 emacs

以下では emacs の使い方を簡単に紹介します。emacs はここで紹介した以外にも多くの機能を持っています。さらに詳しいことを知りたい場合は、詳しくは、emacs の解説書やマニュアルを参照してください。

以下では、次の表記法を使います。

C-x : CTRL キーを押しながら x キーを押す
M-x : ESC キーを押した後で x キーを押す

- emacs の起動

emacs を起動するには、

```
% emacs ファイル名
```

とタイプします。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す。

- 画面

編集するファイルの内容が画面に表示されます。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ます。

- カーソル

ファイルの内容が画面に表示された時に、画面の左上隅の一文字が点滅する四角の中に表示されていますこの四角はカーソルと呼ばれ、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ます。emacs では、カーソルの (左側の) 位置のことをポイント、ドットと呼んでいます。


- カーソルの移動

次のキーをタイプすることにより、文字の書き込まれた範囲で、カーソルの位置を上下左右に動かすことができます。

<u>C-p</u>	↑	上にカーソルを移動します。
<u>C-n</u>	↓	下にカーソルを移動します。
<u>C-b</u>	←	左にカーソルを移動します。
<u>C-f</u>	→	右にカーソルを移動します。

- 文字を書き込む

タイプした文字はカーソルの左側の位置に挿入され、書き込まれていきます。

 をタイプするとそこで改行されます。

- 書き込まれた文字の削除

すでに書き込まれた文字を削除するには、次のようにします。

<u>DEL</u>	カーソルの左側の文字を削除します。
<u>C-d</u>	カーソルの位置の文字を削除します。

- 命令の実行の中止

現在実行中の命令を中止するには、次のようにします。

<u>C-g</u>	現在実行中の命令を中止します。
------------	-----------------

- 編集操作の取り消し

これまでに行なった文字の書き込みや削除の操作を、取り消すには次のようにします。

<u>C-x u</u> または <u>C-</u>	直前に行なった文字の書き込みや削除の操作を取り消します。 取消しを繰り返すことができます。
----------------------------	--

- ファイルへの書き込み、ファイルの読み込み

次のようにすることによって、編集結果を指定したファイルに書き込んだり、指定したファイルの内容を編集画面に読み込んだりすることができます。

<u>C-x C-s</u>	編集結果を現在編集集中のファイルに書き込みます。
<u>C-x C-w</u> Write file: ファイル名 $\boxed{\leftarrow}$	編集結果を指定したファイルに書き込みます。その際ファイル名を答えます。ファイル名を省略すると現在編集集中のファイルが指定されます。
<u>C-x i</u> Insert file: ファイル名 $\boxed{\leftarrow}$	指定されたファイルの内容をカーソルの位置に挿入します。その際ファイル名を答えます。

- emacs の終了
emacs を終了するには次のようにします。

<u>C-x C-c</u>	emacs を終了します。編集結果をまだファイルに書き込んでいないときは、ファイルに書き込むかどうかを聞いてきますので、それに答えます。
----------------	--

2.2 FORTRAN コンパイラ

以下では、FORTRAN コンパイラを用いてソースファイル (プログラムの書かれたファイル) から実行可能ファイルを作る方法について説明します。

- ソースファイル
まず、エディタを使って、FORTRAN のソースファイルを作ります。その際、ソースファイルのファイル名は “.f” で終わっていなければなりません。

例: test.f

- コンパイル、リンク
ソースファイルをコンパイル、リンクするには、

$\boxed{\% \text{ gfortran } \text{ソースファイル名} \boxed{\leftarrow}}$

とタイプします。上の例では

$\% \text{ gfortran test.f} \boxed{\leftarrow}$

とタイプします。gfortran が FORTRAN コンパイラを起動する命令です。これにより、ソースファイルがコンパイルされオブジェクトファイル (機械語に翻訳されたファイル) が作られます。続いて、リンカが起動され、このオブジェクトファイルがすでに用意されているオブジェクトファイルと一つに結合 (リンク) され、a.out という名前の実行可能ファイル (計算機が直接実行できるファイル) が作られます。gfortran はファイル名の最後に “.f” の付いたファイルを FORTRAN のソースファイルであると見なしてコンパイルを行ないます。

gfortran は GNU プロジェクトにより提供されているフリーの FORTRAN コンパイラです。処理系により、他の種類のコンパイラが提供される場合も多く、コマンド名にも色々なものがあります。例えば、f90, frt, ifort, ...。

- 実行

実行可能ファイルを実行するには

```
% 実行可能ファイル名 ↵
```

とタイプします。上の例では

```
% a.out ↵
```

とタイプします。実行可能ファイルは UNIX のコマンドと同様に入出力のリダイレクションを行なうことができます。

- 複数のファイルに分けたとき

FORTRAN の主プログラム、副プログラムが複数のソースファイルに分かれているときは、

```
% gfortran ソースファイル名を空白で区切って並べたもの ↵
```

とタイプします。例えば、

```
% gfortran main.f subroutine.f function.f ↵
```

とタイプします。これにより、コンパイラが起動され、それぞれのソースファイルからオブジェクトファイルが作られます。作られたオブジェクトファイルのファイル名は、ソースファイル名の最後の “.f” を “.o” に変えたものになります。続いて、リンカが起動され、これらのオブジェクトファイルがすでに用意されているオブジェクトファイルと一つに結合 (リンク) され、a.out という名前の実行可能ファイルが作られます。

- コンパイルのみを行なうとき
コンパイルのみを行ない、リンクを行なわないときは、

```
% gfortran -c ソースファイル名を空白で区切って並べたもの ↵
```

とタイプします。

- すでにいくつかのファイルがコンパイルされているとき

```
% gfortran ソースファイル名とオブジェクトファイル名を空白で区切って並べたもの ↵
```

とタイプすることによりソースファイルはコンパイルされ、すでにコンパイルされているオブジェクトファイルとリンクされ、実行型ファイル a.out が作られます。gfortran はファイル名の最後に “.f” の付いたファイルを FORTRAN のソースファイル、ファイル名の最後に “.o” の付いたファイルをオブジェクトファイルと見なします。

- 実行可能ファイルの名前の指定

```
% gfortran ソースファイル名 -o 実行型ファイル名 ↵
```

とタイプすることにより、指定した名前の実行型ファイルが作られます。例えば、

```
% gfortran test.f -o test.x ↵
```

とタイプすると、ソースファイル test.f から実行可能ファイル test.x が作られます。test.x がすでに存在する場合、以前のファイルを削除して新しいファイルを作ります。タイプミスでソースファイルを消すことがあるので、

```
% gfortran test.f ↵
% mv a.out test.x ↵
```

のように mv を用いて、a.out の名前を変えたほうがよいでしょう。

- ソースファイルの拡張子 (山内)

ソースファイルの記述形式には昔から使える固定形式と、FORTRAN90 以降に採用された自由形式があり、現在ではどちらの形式でも使えます。但し、一つのファイルはどちらかの形式で統一しなければなりません。ソースファイルがどちらの形式であるかをコンパイラに指示するには、コンパイラオプションを使って明示的に指定する他、コンパイラによってデフォルトとして設定してあるファイルの拡張子を利用する方法があります。以下のような拡張子ルールを採用しているコンパイラが多いようです。

{ファイル名}.f 或は {ファイル名}.F

では固定形式、

{ファイル名}.f90 或は {ファイル名}.F90

では自由形式です。この場合、例えば、自由形式のソースを”file1.f” に保存して、コンパイルするとエラーになるので対応に注意してください。

“f” と”F” の違いは、コンパイル前に preprocessor を通す場合が大文字、通さない場合が小文字ですが、この実習レベルの段階では気にする必要は無く、どちらを使っても構いません。

2.3 リモートログイン

(注意) 2019 年度までは、目の前の (ローカル) 計算機とは異なる理工学 ITC の共用計算機で実習計算を行っていました。この節はその共用計算機を利用するための説明です。2020 年度からは、ローカルな計算機で実習計算を行う形態に変更しましたので、この節の説明は割愛してもよかったのですが、リモートログインの知識自体は有用なので、そのまま残してあります。(山内)

ローカル計算機から、遠隔 (リモート) 計算機を利用するには、直接そのローカル計算機のディスプレイ、キーボードを使ってログインする必要はありません。理工学 ITC のワークステーション・ルームのワークステーションにログインしてから、ネットワークにつながれた他の UNIX マシンにログインすることができます。このようなログインの方法をリモートログインといいます。

実習室のワークステーションにログインしてから、他の UNIX マシンにリモートログインするには、

```
% slogin ログイン名@ホスト名↵
```

とタイプします。ここで、ホスト名はリモートログインする先の計算機のネットワーク上での名前です。また、ログイン名はリモートログインする先の計算機上でのログイン名です。リモートログインする先の計算機上でのログイン名が現在使っている計算機上でのログイン名と同じ場合には、単に

```
% slogin ホスト名↵
```

とタイプします。例えば、物理学科の physhprx.cm.phys.keio.ac.jp という名前の計算機にリモートログインするには、理工学 ITC の計算機上のログイン名とこれからログインしようとしている計算機上のログイン名とが異なりますから、物理学科の計算機上のログイン名を htakano とするとき


```
% slogin htakano@physhprx.cm.phys.keio.ac.jp ↵
```

とします。slogin はパスワードを含めた通信内容を平文ではなく、暗号化されたデータとして伝達するため、機能的には rlogin, telnet 等のコマンドとほぼ同様ですが、security 上より安全です。

リモートログイン先の計算機から X-protocol を通じて画像を表示する場合には注意が必要です。例えば、リモート計算機で gnuplot を使って、ローカル計算機にグラフを表示させる場合等がこれに相当します。この場合に上記のようにログインして、グラフがローカル計算機に表示されない場合には、そのままの状態では X-protocol の通信が許可されていないと考えられます。この時は、slogin, ssh コマンドにオプションとして "-X" を着けるとよいでしょう。

```
% slogin -X ログイン名@ホスト名 ↵
```

とタイプします。

(重要) 前記のように、授業で計算に使う計算機 ks3.educ.cc.keio.ac.jp(以下 ks3) と、login 端末としての計算機は異なる計算機ですから、通常は当然ながらホームディレクトリ (login 時の directory) は異なり、ファイルの共有などは行われません。ところが、ks3 と ITC の login 端末計算機では、NFS(network file system) を利用して、両者のホームディレクトリが同一のものとなるように設定されています。NFS とは、異なる計算機上でファイルを共有するシステムで、サンマイクロシステム社によって開発されたものです。また login 時のみにホームディレクトリがマウントされるのは automount という機構によるものです。例えば、ITC のワークステーション室のどの端末計算機にログインしても、ls コマンドで見ても、同じファイルが見えます。これも NFS と automount によるものです。ただし、ks3 には、プリント関係のコマンドがインストールされていないので、印刷する時には必ず login 端末から行って下さい。(山内)

2.4 実行の制御

以下では、gfortran などで作られた実行可能ファイルの実行の方法、実行を中止したり、停止したり、再開したりする方法について説明します。

- フォアグラウンドでの実行

前に説明しましたように、gfortran などで作られた実行可能ファイルを実行するには、実行可能ファイル名を UNIX のコマンド名と同様に扱って、プロンプトに対して

```
% 実行可能ファイル名 ↵
```

とタイプします。この場合、実行可能ファイルの実行が終わるまで、プロンプトが返ってきません (UNIX がその端末からは他の命令を受け付けません)。このような実行の方法をフォアグラウンドでの実行といいます。また、フォアグラウンドで実行しているプログラムをフォアグラウンドジョブといいます。

- バックグラウンドでの実行

```
% 実行可能ファイル名 &↵
```

とタイプすることによっても、実行可能ファイルを実行することができます。この場合、実行可能ファイルの実行が終わるのを待たず、プロンプトが返ってきます (UNIX がその端末から命令を受け付けます)。このような実行の方法をバックグラウンドでの実行といいます。また、バックグラウンドで実行しているプログラムをバックグラウンドジョブ、といいます。UNIX がマルチタスクの OS で、1 台の計算機で複数の仕事を同時に行なうことができるため、このようなことが可能です。

- 現在のジョブの状況

現在のジョブの状況を見るには

```
% jobs↵
```

とタイプすることにより、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ます。その際、ジョブごとにつけられた番号はジョブ番号とよばれます。この番号はそのユーザーに対して付けられたものです。また、

```
% ps -l -u ログイン名↵
```

とタイプすることにより、login する前からバックグラウンドで実行されているジョブも含めて、現在実行中、停止中のジョブの状況を見ることが出来る。その際、PID の欄に表示される番号はプロセス番号と呼ばれます。この番号はシステム全体を通して付けられた番号です。S の欄に表示される文字はそのジョブの状態を表します。その文字が R ならジョブが実行中であること、T ならジョブが一時停止中であることを意味します。UNIX では実行されているプログラムをプロセスと呼びます。ps は **p**rocess の略です。

- 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
```

を押すことにより、ジョブの実行を中止することができます。
現在停止中またはバックグラウンドで実行中のジョブで、そのジョブが login 以後に実行したジョブ (jobs で表示される、ジョブ番号のあるジョブ) 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↵
```

とタイプすることにより、ジョブの実行を中止することができます。
現在停止中またはバックグラウンドで実行中のジョブで、そのジョブが login 以前に実行したジョブ (jobs で表示されないジョブ) の場合、プロンプトに対し

```
% kill プロセス番号↵
```

とタイプすることにより、ジョブの実行を中止することができます。

- 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
```

を押すことにより、ジョブの実行を一時停止することができます。
現在バックグラウンドで実行中のジョブで、そのジョブが login 以後に実行したジョブの場合、プロンプトに対し

```
% stop %ジョブ番号↵
```

または

```
% kill -STOP %ジョブ番号↵
```

または

```
% kill -STOP プロセス番号↵
```

とタイプすることにより、ジョブの実行を一時停止することができます。
現在バックグラウンドで実行中のジョブで、そのジョブが login 以前に実行したジョブの場合、

```
% stop プロセス番号↵
```

または

```
% kill -STOP プロセス番号↵
```

とタイプすることにより、ジョブの実行を一時停止することができます。

- 一時停止したジョブの再開

現在停止中のジョブで、そのジョブが login 以後に実行したジョブの場合、

```
% fg %ジョブ番号↵
```

とタイプすることにより、一時停止したジョブをフォアグラウンドで再開することができます。

```
% bg %ジョブ番号↵
```

とタイプすることにより、一時停止したジョブをバックグラウンドで再開することができます。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ます。

現在停止中のジョブで、そのジョブが login 以前に実行したジョブの場合、

```
% kill -CONT プロセス番号↵
```

とタイプすることにより、一時停止したジョブをバックグラウンドで再開することができます。

2.5 プログラムの書き方

ここでは、実際にプログラムを書くときの注意点について述べます。FORTRANの文法については述べませんので、文法書や教科書を参照してください。

2.5.1 プログラミング

プログラミングはだいたい、

1. 問題の分析、解法を選択
2. プログラムの設計
3. 実際にプログラムを書く (コーディング)
4. 検査、誤り (バグ) をとりのぞく (デバッグ)

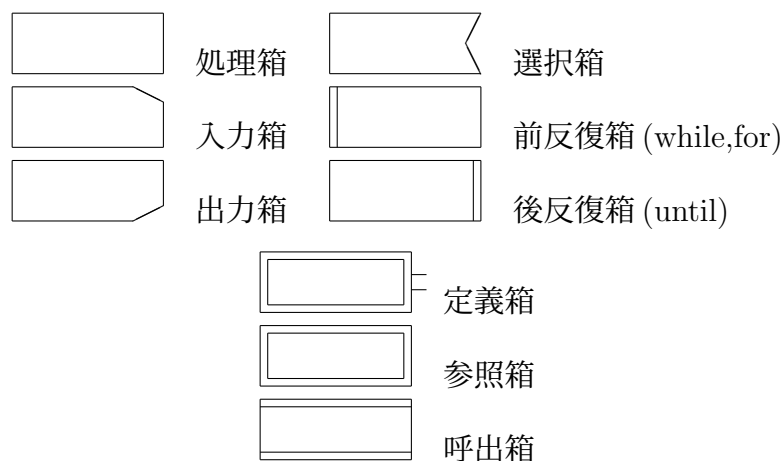
の手順で行ないます。1, 2 のステップは非常に大事ですので、必ず行なうようにしましょう。

2.5.2 PAD

PAD (Problem Analysis Diagram) は、プログラムの構造を非常にわかりやすく表現することのできる図式です。プログラムを設計する際に用いることにより、構造化された見通しの良いプログラムを設計することができます。また、人に自分のプログラムの構造を説明する際などにも、大変役に立ちます。

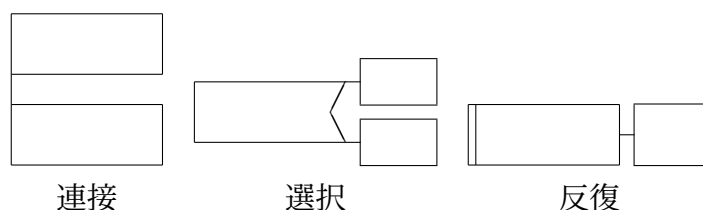
以下では、PAD を簡単に紹介します。フローチャート (流れ図) は使わずに、PAD を使うようにしましょう。付録に PAD の描き方の簡単な説明があります。詳しくは参考書 (川合敏雄, “PAD プログラミング”, (1985, 岩波書店)) を参照してください。

● 構成要素



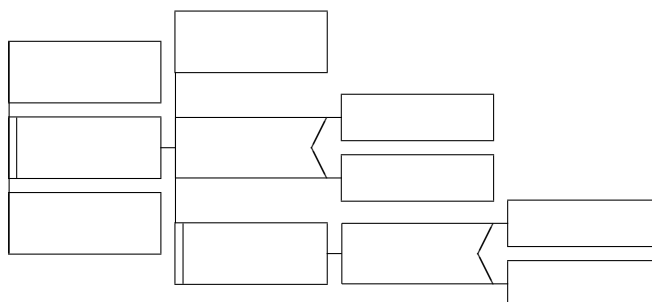
● 基本構造

プログラムは以下の三つの基本構造からなっています。



- プログラムのツリー構造

プログラムは下の例のように、ツリー構造をしています。左から右へ、上から下へとツリーをたどっていくことにより、プログラムの流れを理解できます。



2.5.3 コーディング

ここでは、B.W. Kernighan and P.J. Plauger, 木村泉訳, “プログラム書法”, (1982, 共立出版) からプログラムを書く上での注意点をいくつか紹介します。

- わかりやすく書こう。—うますぎるプログラムはいけない。
- いいたいことを単純率直にいおう。
- わかりやすく書こう。—「効率」のためにわかりやすさを犠牲にしてはいけない。
- かっこを使って誤解を避けよう。
- 混同のおそれのない名前をつかおう。
- FORTRAN の算術 IF 文は使わないこと。
- IF-ELSE を使って、二つのことがらのうち一方だけが起こることをはっきりさせよう。
- プログラムを上から下へと読めるようにしよう。
- 多方向の枝わかれを作るときには IF ... ELSE IF ... ELSE IF ... ELSE ... を使おう。

- 基本的な制御構文を使おう。
- まずわかりやすい疑似言語で書いて、そのあと目的の言語に翻訳しよう。
- 大きいプログラムは小部分ごとに書き、テストするようにしよう。
- 入力 of 妥当性、現実性をテストしよう。
- 入力 that プログラムの限界を決して越えないよう注意。
- すべての変数を、使う前に忘れずに初期設定しよう。
- ひとつ虫をみつけたところで追及をやめてしまてはいけない。
- 一つ違いの誤りに注意。
- プログラムを境界値でテストしよう。
- 速くする前に正しくしよう。
- 意味のある変数名を使おう。
- プログラムの論理構造を反映するような字下げをしよう。

2.5.4 FORTRAN のコーディング

ここでは、FORTRAN のプログラムを書く時の注意点について説明します。

- 実数は常に倍精度実数を使うようにしましょう。
- 暗黙の型宣言は廃止して、変数名は必ず宣言しましょう。以下の宣言で、暗黙の型宣言を無効にできます。

```
implicit none
```

この宣言は FORTRAN90 から正式に採用されたので、現在はどこでも使えるようになりました。古いプログラムには暗黙の型宣言に従ったものが多数あります。古い規約と混乱しないように、変数名の名前の付け方は暗黙の型宣言に従うことを奨めます。(山内)

- DO ループの端末文は必ず END DO 文にしましょう。(山内)
- 異なる DO ループには異なる端末文を使いましょう。
- GOTO 文を使うのは、それを使った方がプログラムがわかりやすくなるときに限りましょう。

- GOTO 文の行き先は CONTINUE 文にしましょう。
- 継続行を示す文字は \$ を使いましょう。
\$ は FORTRAN に許された文字で構文上特別の意味を持たない唯一のものです。
- プログラム中のパラメーターは parameter 文で設定し、プログラム中にマジックナンバーを埋めこまないようにしましょう。
- 倍精度実数の定数は、単精度実数として解釈されないように、1.0d0 のように書きましょう。

2.6 計算機内での数値の表現と誤差

以下では、FORTRAN の整数、倍精度実数が計算機の中でどのように表現されているかの例を紹介します。また実数を用いたときに生じる誤差についても紹介します。

2.6.1 整数の表現

integer*4 で宣言された整数は4バイト、32ビット、1ワードのメモリーを占め、 -2^{31} から $+2^{31} - 1$ の整数を表すことができます。0ビットから30ビットは数を表し、31ビット目は符号ビットになっています。

31	30	...	0
符号	数		

2.6.2 倍精度実数

real*8 または double precision で宣言された倍精度実数は8バイト、64ビット、連続した2ワードのメモリーを占め、おおよそ、

$$-1.79769313486231 \times 10^{308} \sim -2.22507385850721 \times 10^{-308}, 0, \\ +2.22507385850721 \times 10^{-308} \sim +1.79769313486231 \times 10^{308}$$

の実数を表すことができます。倍精度実数は1ビットの符号ビット、11ビットの指数部、52ビットの仮数部からなっており、10進で約16桁の有効数字を持っています。11ビットの指数部は1から2046が-1022から1023までの2進の指数を表しています。52ビットの仮数部は、2進の小数点以下52桁を表しています。小数点以上には1が付いている(規格化されている)ことが仮定されています。指数部が0で、仮数部が0は符号付き0を意味しています。指数部が0で、仮数部が0以外は規格化されていない数を意味します。指数部が2047で、仮数部が0は符号付

き無限大を意味しています。指数部が 2047 で、仮数部が 0 以外は数でない (NAN, not a number) と見なされます。

63	62	...	52	51	...	0
符号	指数部			仮数部		

2.6.3 丸め誤差

仮数部が有限桁であるために、計算機内の実数値は切捨てられ、あるいは丸められます。これによって生じる誤差を丸め誤差といいます。上記の例では、仮数部は 2 進 53 桁ですから、切捨てを行なったときに生じる相対誤差の上限は 2^{-52} となります。この値は浮動小数点の表し方で決る値で、計算機イプシロンと呼ばれます。

2.6.4 情報落ち

大きさの異なる 2 つの実数の加算を行なう場合、小さい方の指数部を大きい方の指数部に合わせて、仮数部の桁をずらしてから加算が実行されます。これによって、小さい方の仮数部の下の方の桁が失われます。これを情報落ちといいます。足しこむときには小さい方から足していきます。

2.6.5 桁落ち

大きさのほとんど同じ 2 つの実数の減算を行なう場合、指数部が等しく、 n 桁の仮数部の上位 k 桁が等しいとすると、減算の結果仮数部の桁数は $n - k$ になってしまいます。これを桁落ちといいます。

2.7 X ウィンドウ

ワークステーションには、普通、ビットマップ・ディスプレイが付いています。ビットマップ・ディスプレイは、画面を格子状に区切った画素からなり、個々の画素の色や明るさを制御することにより画像を表示することのできるディスプレイです。普通 UNIX では、X ウィンドウ・システムとよばれる、ビットマップ・ディスプレイ上でグラフィカル・ユーザ・インターフェイスを実現するためのソフトウェアが使えます。これにより、ウィンドウと呼ばれる、図形や文字を表示することのできる長方形の領域を生成したり、動かしたり、大きさを変えたりすることができます。ウィンドウは重なり合うこともできます。X ウィンドウを用いたソフトウェアでは、画面上での位置を指定したり、いろいろな指示を与えるために、3 つのボタンのついたマウスと呼ばれる装置を用います。

a. X ウィンドウの起動

理工学 ITC のワークステーションでは、プロンプトに対して、

```
% startx ↵
```

とタイプします。すると、理工学 ITC の設定では、3つのウィンドウ (長方形の領域) が表示されます。各ウィンドウの上部には、左からアイコン化ボタン、タイトルバー、リサイズボタンがあります。1つのウィンドウはコンソール・ウィンドウと呼ばれ、login という名前が、他の2つのウィンドウには kterm という名前がいています。マウスを動かすことにより、マウスカーソルと呼ばれる印が画面上で動きます。

b. ウィンドウの選択

マウスカーソルをウィンドウ内に移動させることにより、そのウィンドウを端末として入力、出力を行なうことができます。そのウィンドウをアクティブ・ウィンドウと呼びます。通常は、コンソール・ウィンドウ以外のウィンドウを選択して作業します。

c. ウィンドウの移動

タイトルバーの上にマウスカーソルを移動し、マウスの左ボタンを押しながらマウスを動かすと、マウスカーソルの移動に伴ってウィンドウが移動します。その際、そのウィンドウは一番上になります。

d. ウィンドウの上下関係の変更

タイトルバーの上にマウスカーソルを移動し、マウスの中ボタンを一回押す (クリック) することにより、そのウィンドウを一番上にしたり、ウィンドウを一番下に移動します。

e. ウィンドウの大きさの変更

リサイズボタンの上にマウスカーソルを移動し、マウスの左ボタンを押しながらマウスを動かすと、マウスカーソルがウィンドウの枠を横切った後、マウスカーソルの動きに合わせて枠の大きさが変化します。

f. X ウィンドウの終了

kterm ウィンドウではマウスカーソルをそのウィンドウに合わせて、

```
% exit ↵
```

とタイプすることにより、そのウィンドウを閉じる (消す) ことができます。同様にして、最後に login と名の付いたコンソール・ウィンドウを消すことにより、X ウィンドウ・システムを終了することができます。

2.8 gnuplot

gnuplot というソフトウェアを利用することにより、X ウィンドウ上で簡単にグラフを描くことができます。

a. gnuplot の起動

starx で X ウィンドウ・システムを起動し、アクティブ・ウィンドウ上のプロンプトに対して、

```
% gnuplot↵
```

とタイプします。すると、gnuplot のプロンプト `>` が表示され、gnuplot が命令を待っている状態になります。

b. gnuplot の終了

gnuplot のプロンプトに対し、

```
> quit↵
```

とタイプすると、gnuplot が終了し、UNIX のプロンプトに戻ります。

c. グラフのデータのファイル

ここでは、 x の数値、 y の数値が空白で区切られて 1 行に 1 組ずつ書き込まれたファイルを考えます。

x_1	y_1
x_2	y_2
x_3	y_3
\vdots	\vdots

d. グラフのプロット

gnuplot のプロンプトに対し、

```
> plot "データファイル名" with lines↵
```

とタイプすることにより、新しいウィンドウが開き、データファイル内の x , y の値の組が順に線で結ばれてプロットされます。

```
> plot "データファイル名" with points↵
```

とすると、点でプロットされます。

"データファイル名" with lines

の部分は“,”で区切って並べることにより複数のファイルを同時にプロットすることができます。

e. プロットするグラフの追加

```
> plot "データファイル名 1" with lines↵
```

としてグラフをプロットした後で

```
> replot "データファイル名 2" with points↵
```

とすることにより、このデータのプロットを追加することができます。

f. プロット範囲の指定

```
> set xrange [ x の下限 : x の上限 ]↵
```

```
> set yrange [ y の下限 : y の上限 ]↵
```

などとすることにより、プロットの範囲を指定できます。また、次のように plot コマンドに直接指定しても構いません。

```
> plot [ x 下限:上限 ][ y 下限:上限 ]"データファイル名"↵
```

g. 対数軸

```
> set logscale x↵
```

```
> set logscale y↵
```

などとすることにより、それぞれの軸を対数軸にすることができます。

```
> set nologscale x↵
```

```
> set nologscale y↵
```

などとすることにより、それぞれの軸を普通の軸に戻すことができます。

h. 再プロット

プロットした後で、範囲のような設定を変えたような場合、

```
> replot↵
```

とすることにより、新しい設定で再プロットします。

i. プロットするデータの指定

データファイルにいくつかのデータが並んでいて、n1 番目のデータを横軸に、n2 番目のデータを縦軸にプロットするには

```
> plot "データファイル名" using n1:n2↵
```

と指定する。このオプション省略時のデフォルトは n1=1, n2=2 です。(山内)

j. ポストスクリプト・ファイルへの出力

グラフをプロットした後で

```
> set term postscript↵
> set output "ポストスクリプト・ファイル名"↵
> replot↵
```

とすることにより、そのグラフがポストスクリプト形式でファイルに出力されます。

```
> set term x11↵
```

とすると、X ウィンドウ上に再びグラフを描くことができます。

k. ポストスクリプト・ファイルの表示

```
% ghostview ポストスクリプト・ファイル名↵
```

とタイプすることにより、新しいウィンドウが作られポストスクリプト・ファイルが X ウィンドウ上に表示されます。ghostview コマンドを終了するには、新しく作られたウィンドウを選択して q とタイプします。そのほか、マウスを使っていろいろな操作を行うことができます。ghostview が入っていない計算機には、

```
% evince ポストスクリプト・ファイル名↵
```

をためしてみてください。evince は、postscript file 以外にも、pdf 等のいくつかのファイル形式を表示することができます。

1. ポストスクリプト・ファイルの印刷

理工学 ITC のワークステーション・ルームでは、

```
% lpr ポストスクリプト・ファイル名↵
```

とタイプすることにより、**ポストスクリプト・ファイル**が印刷されます

(注意) lpr コマンドで印刷できるものは、ポストスクリプトファイルのみです。プログラム等のテキストファイルはそのままでは印刷できません。ポストスクリプトに変換してから印刷しましょう。

印刷についての詳細は、理工学 ITC のホームページを参照してください。

2.9 計算物理学実習に使用する計算機

以下の計算機を利用することができます。

ワークステーション室にある計算機 (login 端末 PC)

遠隔授業で許可された計算機 (個人毎に割り当て)

これら計算機のホームディレクトリは ITC の login 端末 PC と共通になっています。また、パスワードも共通になっています。これらの計算機でパスワードを変更するには、

```
% yppasswd↵
```

とタイプします。passwd ではないので注意してください。

2.10 ファイル転送

(注意) この節も 2020 年度からの共用計算機使用廃止に伴い、必ずしも必要ではなくなりましたが、ファイル転送の概念は重要なので残してあります。そのつもりで読んでください。(山内)

ftp というソフトウェアを利用することにより、計算機間でファイルの転送を行なうことができます。ks3.educ.cc.keio.ac.jp では、端末 PC とホームディレクトリが共有されているので、ファイル転送は必要ありませんが、一般には頻繁に使用することになりますので、覚えておいて下さい。

a. ftp の起動

今ログインしている計算機のプロンプトに対して、

```
% ftp リモートホスト名
```

とタイプします。ここで、リモートホスト名とは、ファイルのやりとりを行なう相手の計算機の名前です。これに対し、今ログインしている計算機をローカルホストといいます。すると、ログイン名とパスワードを順番にきいてきますので、相手の計算機のログイン名とパスワードを入力します。すると、ログインに成功したか、失敗したかが表示された後、ftp のプロンプト ftp> が表示され、ftp が命令を待っている状態になります。

b. ftp の終了

ftp のプロンプトに対し、

```
ftp> quit
```

とタイプすると、ftp が終了し、UNIX のプロンプトに戻ります。

c. ftp のコマンド

次のようなコマンドを使って、ファイルの転送を行ないます。

get ファイル名	リモートのファイルを ローカルのカレントディレクトリにコピーします。
put ファイル名	ローカルファイルを リモートのカレントディレクトリにコピーします。
cd ディレクトリ名	リモートのカレントディレクトリを変更します。
lcd ディレクトリ名	ローカルのカレントディレクトリを変更します。
mkdir ディレクトリ名	リモートにディレクトリを作ります。
!mkdir ディレクトリ名	ローカルにディレクトリを作ります。
pwd	リモートのカレントディレクトリを表示します。
!pwd	ローカルのカレントディレクトリを表示します。
ls	リモートのカレントディレクトリのファイルを表示します。
!ls	ローカルのカレントディレクトリのファイルを表示します。

- d. 最近ではセキュリティ上の問題から ftp が使えないケースが増えて来ました。このような場合には、ftp の security 強化版である sftp が使えるか試してみてください。もしくは ssh、slogin 等のコマンドが使えるホスト間では scp でファイルの送受信を行えます。書式は以下のようになります。現在使っている計算機のファイル file1 を、ホスト名 host2 の計算機上のユーザー user2 にアクセス権のあるファイル file2 へコピーする場合には

```
% scp file1 user2@host2:file2 ↵
```

逆に、ホスト名 `host2` の計算機上のユーザー `user2` にアクセス権のあるファイル `file2` を現在使っている計算機のファイル `file1` へコピーする場合には

```
% scp user2@host2:file2 file1 ↵
```

となります。尚、計算機 `host2` 上でのユーザー名 `user2` が、現在使用している計算機上でのユーザー名と同じ場合には”`user2@`” の部分を省略できます。
(山内)

2.11 FORTRAN でのファイルの読み書き

計算物理学実習で使用する計算機上の FORTRAN では、次のようにファイルを読み書きします。

2.11.1 open 文でファイル名を指定する場合

FORTRAN である装置番号に特定のファイルを対応させるには `open` 文を使います。

```
open(装置番号, file='ファイル名')
```

この対応をやめるには `close` 文を使います。

```
close(装置番号)
```

例えば、

```
open(9,file='test1.dat')
write(9,*) 1
close(9)
open(9,file='test2.dat')
write(9,*) 2
close(9)
```

とすることにより、ファイル `test1.dat` には 1 が、ファイル `test2.dat` には 2 が書かれます。

2.11.2 open 文でファイル名を指定しない場合

ks3.educ.cc.keio.ac.jp 上の FORTRAN では、open 文でファイル名を指定しないと、装置番号 5 は標準入力に、装置番号 6 は標準出力に、装置番号 7 は標準エラー出力に対応づけられています。

read(5, ...), read(*, ...)	は標準入力から入力を受けます。
write(5, ...)	はエラーになります。
write(6, ...), write(*, ...)	は標準出力に出力します。
read(6, ...)	はエラーになります。
write(7, ...)	は標準エラー出力に出力します。
read(7, ...)	はエラーになります。

これら以外で open 文でファイル名が指定されていないと、装置番号 XX ($XX=01, 02, 03, 04, 08, \dots, 99$) に対して read か write が初めて行なわれた時点で装置番号 XX に `ftnXX` という名前のファイルが対応づけられます (open されます)。例えば、

```

      read(9,*)_n
      write(11,*)_n

```

とすることにより、ファイル `ftn09` から変数 `n` が読まれて、その値が `ftn11` に書き込まれます。

2.12 課題

- 2-A.** 倍精度実数で $1+2^{-n}$ を計算し、それが倍精度実数の 1 より大きいような最大の整数 n を求め、 n と 2^{-n} (計算機イプシロン) を出力せよ。
- 2-B.** 体積 V の箱の中の電磁波が温度 T の熱平衡にあるとき、角振動数 $\omega \sim \omega + d\omega$ の単位体積当たりの電磁波のエネルギーを $u(\omega, T)d\omega$ とする。Planck の輻射式

$$u(\omega, T) = \frac{\omega^2}{\pi^2 c^3} \frac{\hbar \omega}{e^{\beta \hbar \omega} - 1}$$

のグラフを描け。ここで、 $\beta = \frac{1}{k_B T}$ で k_B は Boltzmann 定数。さらに、Rayleigh-Jeans の輻射式

$$u(\omega, T) = \frac{\omega^2}{\pi^2 c^3} \frac{1}{\beta}$$

Wien の輻射式

$$u(\omega, T) = \frac{\omega^2}{\pi^2 c^3} \hbar \omega e^{-\beta \hbar \omega}$$

のグラフも同時に描き比較せよ。グラフは、無次元化して、 $\beta^3 \pi^2 \hbar^2 c^3 u(\omega, T)$ 対 $\beta \hbar \omega$ のグラフを描くこと。

2-C. 原子数 N の固体の定積比熱 C_V に対する Debye の式は

$$C_V = 3Nk_B \cdot 3 \left(\frac{T}{\Theta_D} \right)^3 \int_0^{\frac{\Theta_D}{T}} \frac{x^4 e^x}{(e^x - 1)^2} dx$$

で与えられる。ここで、 Θ_D は Debye 温度。無次元化して、 $C_V/(3Nk_B)$ 対 T/Θ_D のグラフを描け。Dulong–Petit の古典的値

$$C_V = 3Nk_B,$$

$T \ll \Theta_D$ の低温で期待される振舞い

$$C_V = (3Nk_B) \frac{4\pi^4}{5} \left(\frac{T}{\Theta_D} \right)^3$$

と比較せよ。数値積分は、台形公式、シンプソン公式などを用いよ。刻み巾を変えて数値積分の誤差について検討せよ。

▷ 台形公式、シンプソン公式

$i = 0, 1, 2, \dots, n$, $h = (b - a)/n$ に対し、

$$x_i = a + i * h$$

とする。このとき、台形公式は

$$\int_a^b f(x) dx \simeq \frac{h}{2} (f_0 + 2f_1 + 2f_2 + \dots + 2f_{n-1} + f_n)$$

で与えられる。シンプソン公式は n を偶数として

$$\int_a^b f(x) dx \simeq \frac{h}{3} (f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + \dots + 2f_{n-2} + 4f_{n-1} + f_n)$$

で与えられる。ただし $f_i = f(x_i)$ とする。