3 UNIX システムとプログラミング言語

3.1 X Window System

キーボードと文字表示画面による計算機の操作方法を CUI(Character User Interface) というのに対し、マウス等のポインタ装置とグラフィックス画面による操作方法を GUI(Graphical User Interface) という。Microsoft 社の Windows や Apple 社の MacOS は、GUI を前面に押し出した OS である。

UNIX におけるグラフィックス表示の標準は**X Window System** (以下では単に X と呼ぶ) である。これは 1984 年に米国マサチューセッツ工科大学 (MIT) が、IBM や DEC(Compaq を経て現在 HP) の協力を得て開発したものである。

X の特徴の 1 つが、クライアント・サーバーモデルによるネットワークに対する透過性である。

X の GUI の操作性には、Windows や MacOS の GUI とは異なる点が幾つかあるので、ここでまとめておく。まず、X で使われるマウスには標準的にボタンが 3 つある[1]。それぞれの役割は、文字列のカット&ペースト (切り取りと張り付け) については、

左ボタン 領域の選択開始 (ボタンを押したままドラッグしても良い)

右ボタン 領域の終点を指定 (ドラッグした場合はボタンを離す)

中ボタン 選択領域のペースト

であり、スクロールバーにおける操作は、

左ボタン ウインドウを下方へスクロール

右ボタン ウインドウを上方へスクロール

中ボタン ボタンを押し続ける事により、ウインドウを掴んだままスクロール

という動作が標準的である。

本講義で使用する `kterm` や `mule` といった伝統的な X アプリケーションには、スクロールバーも無い簡素な (不愛想な) ウインドウを表示するものが多い。Ctrl キーや Shift キーとマウスボタンを同時に押す事によって設定変更メニューがプルダウンされるので、色々試してみて欲しい。例えば、スクロールバーの表示・非表示を切替えたり、文字の大きさを変えたりといった事が出来る。

3.2 シェル–tcsh–

UNIX の伝統の 1 つに、「小さくて簡単 (従って動作も速くデバックも容易) なプログラム (コマンド)のみ用意」し「それらを組み合わせる事により複雑な処理を実現したり処理を自動化する」がある。この考え方は、基本的にはプログラミングにおいても有益である。今日流行のグラフィカルなプログラムは、初心者にとっては敷居が低いが、動作が緩慢だったり、エラーを引き起こしたり[2]、ユーザーの習熟度に応じて作業効率が必ずしも向上しない等の問題がしばしば起こる。

[1] 伝統的に Apple 社の Macintosh では 1 つボタンで、Windows もかつては 2 つボタンが主であった。これらの機種では、キーボードのあるキーを押しながらクリックする事や、2 つのボタンを同時にクリックする事で他のボタンをエミュレートするものがある

[2] 最悪の場合は、OS を道連れにしてフリーズしてしまう。

学生諸氏も、自宅の Windows のアプリケーションでこのような問題に遭遇したことがあるであろう。UNIX のコマンドは、始めは敷居が高く感じられるかも逸れないが、習熟した後の作業効率や応用度の高さでは比肩するものが無い。また UNIX には多くの派生バージョンが存在するが、基本的なコマンドの使い方は同じであるので、将来他の UNIX システムを利用する際にも有効であるので是非マスターして欲しい。

UNIX では、ユーザーの入力するコマンドを解釈・実行するプログラムは**シェル**と呼ばれる。シェルも単なるアプリケーションプログラムの 1 つであるので、ユーザーは好みに応じて色々なシェルを使い分ける事ができる。本講義で用いる情報基盤センターの UNIX システムでは、使い勝手が良く、C 言語に似たスクリプトが書ける `tcsh` というシェルが標準となっている。`kterm/xterm` を起動すると自動的に `tcsh` が起動するように設定されているので、改めて `tcsh` を起動する必要は無い。

実際の作業に際して最低限知っておいた方が良いと思われる事を以下にまとめる。[ ]で囲まれている文字はキーボードのキーを表し、[Ctrl-p]等と記されているのは[Ctrl]キーと[p]を同時に押す事を意味する。また[↵]は[return]を表す。

コマンドヒストリー

繰り返し同じようなコマンドを入力するとき、あるいはコマンドを間違えたときは、コマンドを最初から入力するよりも、以前に入力したコマンドを呼び出してそれを修正する方が効率的である。

`tcsh` では実行されたコマンドは、ある程度の行数まで記憶されており (コマンドヒストリーという)、カーソル・キーの[↑]と[↓]、または[Ctrl-p][3]と[Ctrl-n][4]で呼び出した後、編集して実行させる事ができる。

他によく使われる機能として、直前に実行したコマンドを再実行する、

```
% !!
```

であるとか、先頭が `l` で始まるコマンドのうち、最も最近実行したものを再実行する、

```
% !l
```

がある。

コマンドの編集

コマンドのキー入力においては、カーソル移動キー[←](または[Ctrl-b])、[→](または[Ctrl-f])、[Ctrl-a](行頭に移動)、[Ctrl-e](行末に移動)、[Ctrl-d](カーソル位置の文字を消去)、[Ctrl-k](カーソル位置から行末まで消去) 等による編集が可能である。

前述のコマンドヒストリーも含め、キーの割り当ては全て、3.5 節で説明する `mule` と呼ばれるエディタと共通になっているので、一度覚えてしまうと非常に便利に使える。

コマンド・ファイル名補完

コマンド名やファイル名の最初の数文字をタイプした後、[Tab]または[Ctrl-i]を押すと、入力済みの文字で始まるコマンドやファイルが他に無い場合には、残りの部分を補ってくれる。この機能は**補完機能**と呼ばれ、タイプ量を減らすのに非常に役立つ。例えば、

[3] Previous の “p”
[4] Next の “n”

```
% gnu[Tab]
```

とタイプすると、端末画面には、

```
% gnuplot
```

というように、残りの...plot の部分が補完されて表示される。

最初にタイプした文字が少なすぎて候補が1つに絞れない場合、補完はされず警告音 (ブザー) がなる。この時は、[Ctrl-d]を押すと候補の一覧が表示される。例えば、

```
% gn[Ctrl-d]
```

とタイプすると、端末画面には、

```
% gnect gnibbles
```

というように、gn... ではじまるコマンドの一覧が表示される。

プログラムの停止

作成したプログラムが無限ループに入ってしまった場合など、プログラムの実行を強制的に停止させたい時は[Ctrl-c] を押せばよい (但し、既存コマンドの中には例外的に止まらないものもある)。

ジョブ制御 (バックグラウンド起動)

起動後に独立したウインドウを開いて動作するプログラム (例えば mule) は、一旦起動すると文字端末 (kterm) は用済みであるので、バックグラウンドで動作させる事により端末を解放した方が、続いてコマンドを実行できるので便利である。

ここで、**フォアグラウンド**と**バックグラウンド**という言葉がでてくるが、キーボードの入力を伝えるジョブをフォアグラウンドのジョブといい、それ以外のジョブはバックグラウンドにいるという。

例えば mule というエディタをバックグラウンドで起動するには、

```
% mule &
```

と、コマンド行の末尾に&を付ければ良い。この場合、mule を起動した端末では、続けてコマンドを入力・実行する事が出来る[5]。

他方&を付けずに mule を起動した場合、kterm のウインドウをアクティブにしてもキー入力は受け付けない[6]。これは mule がフォアグラウンドで動作しているためだが、バックグラウンドに移すためには、

1. mule を起動した端末 (kterm) をアクティブにする
2. 端末で[Ctrl-z]を入力して**サスペンド (中断)** する
3. 端末で

[5] 一般に、mule を起動した段階で、アクティブなウインドウ (キーボードの入力を受け付けるウインドウ) は kterm から mule に移っているが、kterm を選択しアクティブにすれば入力が可能となる。

[6] キー入力すると画面には表示されるが、シェルは受け取っていない。

```
% bg
```

というコマンド (BackGround) を実行すれば良い。

他に、**リダイレクト**と**パイプ**と呼ばれる重要な機能がある、これについては 3.4 節を参照の事。

tcsh は多くの機能を持ち、ある種のプログラム (シェルスクリプトと呼ばれる) を書く事も可能である。より詳しい事は man tcsh としてオンラインマニュアルを参照するなり、インターネット上の情報を活用するなりして欲しい。

3.3 マニュアル

3.3.1 歩くマニュアル

UNIX という環境に親しんだり、FORTRAN・C といったプログラミング言語を身に付けるのは、異国で生活し新しい外国語を学ぶのに似ている。良いお手本に従って反復したり、上手に使いこなしているひとの「まね」をする事が、参考書やマニュアルを読破するより遥かに上達の早道である。その意味で、身の回りのエキスパート、すなわち**歩くマニュアル**は有効に活用しよう[7]。

3.3.2 man

歩くマニュアルは有効ではあるが、全てを聞いていたのでは何も身に付かないので、自分で調べる努力も重要である。

コマンド名を知っていて、その使用方法を詳しく知りたい場合は、以下の例のようにすると画面にマニュアルが表示される。

```
% man cat↵
```

逆に、コマンドの働きを示すキーワードを指定して、そのキーワードに関連したコマンド名の一覧を表示させるには、例えば、

```
% man -k compare↵
```

あるいは

```
% apropos compare↵
```

とすればよい。

3.4 基本コマンド

以下に、普段の作業でよく使われるコマンド、知っていると便利なコマンドをまとめる。大部分はファイルやディレクトリの操作に関するものである。ディレクトリとファイル、またはディレクトリ同士は/によって区切られる。ファイルやディレクトリの名前には、/以外の任意の文字を使う事ができるが[8]、後のトラブルを避けるため、アルファベットと数字、及び_(underscore)のみからなる、数文字から十数文字の長さの名前をつける方が無難である[9]。なお、アルファベットは**大文字と小文字が区別される**事に注意せよ。

[7] 人に教えるという事は、理解を深める上で非常に有益であるので、歩くマニュアルの方にも利益がある。
[8] 空白や漢字も使える。
[9] UNIX のコマンド (ls や cp 等) は避けた方が無難である。

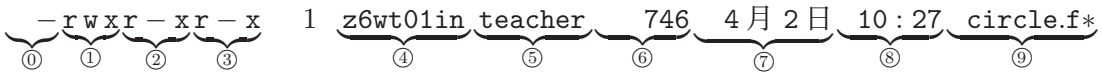
ファイル名やディレクトリに関する特別な約束がいくつかある。.(ピリオド) で始まる名前のファイルは、通常は表示されない**隠しファイル**であり、各種のコマンドの設定を記録するために使われる[10]。ところがややこしいことに、単独のピリオド "." は現在の作業ディレクトリ (pwd で表示されるもの) を指し、二つのピリオド ".." は一つ上の階層のディレクトリを指すという約束がある。また、ピリオドは**拡張子**の区切りにも使用される[11]。拡張子はファイルの種類や属性を端的に表すために用いられるが、mule やコンパイラを含む幾つかのプログラムでは、動作がファイルの拡張子に依存するため注意が必要である。

ファイルやディレクトリの名前を表すのに、*(任意の長さの任意の文字列) や?(任意の 1 文字) 等の、いわゆるワイルドカードを使う事ができる。この解釈はシェルが行っており、他にも [a-z](アルファベット小文字 1 文字を代表) 等が使える。詳しくは man tcsh の Filename substitution(ファイル名置換) の説明を参照せよ。ファイル名置換でもう一つ重要な文字に~ (チルダ) がある。~ がディレクトリ名として使われた時はユーザーのホームディレクトリ (例えば、ユーザー名が z6wt01in の場合、/home/teacher/z6wt01in というディレクトリ) を意味する[12]。例えば、~/sample.f は /home/teacher/z6wt01in/sample.f に等しく、この記法は mule/emacs 等のエディタでも使われている。

ls : ディレクトリの表示 (list)

```
ls *.c   (.c で終わる全てのファイル表示)
ls -a    (隠しファイルを含む全てのファイル表示)
ls -l    (ファイルの付随情報を表示)
```

-l オプションを付けて実行した場合、例えば、



と表示されるが、それぞれの部分の意味は以下の通り。

⓪ ファイル種別 (-:通常ファイル、d:ディレクトリ、b,c:デバイスファイル、等)

① ユーザー (user) のアクセス設定

② グループ (group) のアクセス設定

③ それ以外 (others) のアクセス設定

④ ファイルを所有するユーザー名

⑤ ファイルを所有するグループ名

⑥ ファイルの大きさ (バイト単位)

⑦ ファイルを最後に変更した日

⑧ ファイルを最後に変更した時間

[10]例えば tcsh は起動時に.tcshrc という名前を読み込み設定がなされる。

[11]Windows でも、通常は拡張子は表示されないが、ファイルの種類の指定に使われており、この拡張子によりアイコンが変化する。

[12]学生諸氏の場合は、teacher の部分は入学年度に応じて 04nen 等となる。

⑨ ファイル名 (実行ファイルは最後に*が付く)

アクセス設定は、r が読み出し許可、w が書き込み許可、x が実行許可 (ディレクトリの場合は移動許可) を表し、rwx の並び (セット) で user/group/others いずれかに対するアクセス設定をなす。-となっている場合は、該当する許可が無い事を表す。

アクセス設定の変更は、次の chmod で行う。

chmod : アクセス設定の変更 (change mode)

```
chmod g+rw test.dat  (group に対して test.dat の読み書き (rw) を許可 (+)
chmod o-rx test.dat  (others に対して test.dat の読み (r) と実行 (x) を禁止 (-)
```

対象者 (u:user,g:group,o:others,a:all) に対して、アクセス (r:read,w:write,x:execute) を許可 (+) または禁止 (-) する。ただし許可の場合は、上の階層のディレクトリも x になっていなければ意味をなさない。

cat : ファイルの表示・連結 (concatenate)

```
cat sample1.dat                    (samle1.dat の内容を表示)
  1  A
  2  B
cat sample2.dat                    (samle2.dat の内容を表示)
  3  C
  4  D
cat sample1.dat sample2.dat        (二つの内容を連結して表示)
  1  A
  2  B
  3  C
  4  D
```

上の例を見ても、ファイルの内容の表示以上に何の役に立つかピンとこないかも知れない (内容を見るだけなら mule 等のエディタでも見れる)。UNIX では、画面出力を**標準出力**(stdout)[13]、キーボード入力を**標準入力**(stdin)[14]という**特殊なファイル**と位置付けている。従って、入力元・出力先を通常のファイルに変更 (**リダイレクト**という) する事が可能である。例えば以下のように標準出力をファイルにリダイレクトすると、二つの内容を連結したものが sample3.dat に保存される。

```
cat sample1.dat sample2.dat > sample3.dat
```

ここで、“>” は出力先を標準出力 (画面 stdout) からファイル (sample3.dat) にリダイレクトする事を表す。

また、パラメータをキーボードから読み込んで実行するプログラム prog1 があるとして、

```
prog1 < sample3.dat
```

または

[13]standard output.
[14]standard input

```
cat sample3.dat | prog1
```

とすれば、prog1 はパラメータを sample3.dat から読み込んで実行する[15]。二番目の例では、

cat の標準出力を、prog1 の標準入力に渡す

という事が行われている。この仲介が “|” で行われており**パイプ**と呼ばれる。
リダイレクトには追加モードもあり、

```
cat sample2.dat >> sample1.dat
```

とすると、sample1.dat も末尾に sample2.dat の内容が付け加えられる (つまり、sample1.dat は sample3.dat と等しくなる)。
キーボードからの入力を sample.dat に保存するには、

```
cat << EOF > sample.dat
  11  101
  12  102
EOF
```

とすると、区切りを指定する EOF [16]に出会うまでの全ての文字を標準入力から受け付けて、ファイル sample.dat に書き出す[17]。cat を含む幾つかのプログラムは、単独の-を標準入力と解釈するので、上記と同じ事は、

```
cat - > sample.dat
  11  101
  12  102
Ctrl-D
```

としてもよい[18]。最後の Ctrl-D は UNIX では一般にファイルの終了を意味する特殊文字である[19]。
パイプやリダイレクトは、「小さくて簡単なプログラム (コマンド)のみ用意し、それらを組み合わせる事により複雑な処理を実現したり処理を自動化する」という UNIX の考え方と深く結び付いている。積極的に利用して、UNIX を非常に便利かつ効率的に使用して欲しい。

cp : ファイルのコピー (copy)

```
cp file1 file2                  (ファイル file1 をファイル file2 にコピー)
cp file1 file2 file3 dir1       (file1-file3 の三つのファイルを、
                                 ディレクトリ dir1 以下にコピー)
cp -r dir1 dir2                 (dir1 以下の全てを dir2 にコピー。
                                 -r は Recursive(再帰的) を表す。)
cp -i file1 file2               (file2 が上書きになる時は確認を求める)
```

[15]同じ入力パラメータに対して、プログラムを少しずつ変えながら実行する場合に、入力の手間が省ける。
[16]End Of File
[17]EOF は実際は何でも構わないが、EOF が良く用いられる。
[18]cat は元々標準入力を受け付けるので、この場合は-は省略できる。
[19]^D とも書かれる

最初の二つの例では、file2 が既に存在していたり、dir1 以下に file1 が存在していても、cp は何の警告も出さずファイルを上書きする事に注意[20]。これを避けるには、-i オプションを付けると良い。上書きになる場合には確認を求めて来るようになる。慣れないうちは-i を付けるように習慣付けるとよい。

cp : ファイルの移動・改名 (move)

```
mv file1 file2                (ファイルの名前を file1.dat から file2 に変更)
mv file1 file2 file3 dir1     (file1-file3 の三つのファイルを、
                               ディレクトリ dir1 以下に移動)
```

標準では警告無しに上書きをするので、-i オプションを付ける事が推奨される点は cp と同じである。

rm : ファイルの削除 (remove)

```
rm *.f                        (.f で終わるファイル全てを削除)
rm -i a*                      (a で始まる全てのファイルを確認しながら削除)
rm -r dir1                    (dir1 以下を再帰的に削除)
```

rm は本当にファイルを削除する (Windows や MacOS のように、一旦ゴミ箱に移動し、後で改めて本当に削除する訳ではない) ので、慎重に使用すべきである。慣れないうちは-i オプションを付けるように習慣付けた方が良いであろう。

ワイルドカードを使う時は特に注意が必要である。例えば一番目の例で、「うっかり*と .f の間にスペース (空白) を入れてしまった」場合に何が起こるか想像して欲しい。この他のありがちな間違いとして、. で始まる隠しファイルを削除するつもりで

```
rm .*
```

を実行してしまう事である。.*は.、つまり現在の作業ディレクトリにもマッチするので、rm . に等しい事が起こってしまう。

ワイルドカードを使う場合には、tcsh の展開機能を使い、マッチするファイルの一覧に変換してからコマンドを実行する事が推奨される。具体的には、一番目の例を実行する場合、

```
rm *.f Ctrl-x *
```

と、(リターンを押す前に) Ctrl キーと x を同時に押してから*を押すと、

```
rm file_open.f gauss.f runge_kutta.f
```

等と展開されるので、ファイルの一覧を確認してからリターンを押してコマンドを実行すれば良い。

mkdir : ディレクトリの作成 (make directory)

```
mkdir dir1                    (dir1 というディレクトリを作成する)
```

ディレクトリは Windows や MacOS でのフォルダに対応する。ディレクトリをうまく活用して、ファイルを整理して欲しい。

rmdir : ディレクトリの削除 (remove directory)

[20] UNIX は元来寡黙な OS である。

```
rmdir dir1                    (dir1 というディレクトリを削除する)
```

ただし、ディレクトリの中身が空になっていないと削除できない。よって、rm -r dir1 の方が実用性が高い。

pwd : 現在の作業ディレクトリの表示 (present working directory)

```
% pwd
/home/teacher/z6wt01in
```

cd : 現在の作業ディレクトリの変更 (change directory)

```
cd dir1                       (ディレクトリ dir1 へ移動)
cd ..                         (一つ上の階層のディレクトリに移動)
cd                            (ユーザーのルートディレクトリ (~) に移動)
```

類似のコマンドに pushd と popd がある。使いこなすと便利であるので、man で調べて欲しい。

less,diff,grep,find : その他の有用なコマンド

```
less file1                    (file1 の閲覧。 q で終了。
                               a.out | less 等、パイプで使うと便利)
diff file1 file2              (file1 と file2 の比較)
grep string file1             (文字列 string を file1 の中から検索。
                               大文字・小文字を区別しない時には-i を付ける)
find dir -name pattern        (ディレクトリ dir 以下の pattern にマッチする
                               名前を持つファイルを検索する。
```

UNIX のコマンドはそれ自身意味があるが、パイプやリダイレクションを併用した場合にその真価を発揮する。例えば、

```
find .  -name '*.f' | xargs grep -n real
```

は、現在の作業ディレクトリ . 以下の全てのディレクトリから拡張子.f を持つファイルを探し、その中で real という文字列を含むものを探し、その行を行番号と共に表示するという一連の作業を行う (シェルに*を解釈させないために引用符'...' で*.f を囲っている事に注意。*.f と書いても良い)。xargs や find の他のオプションについては、man で調べて欲しい。

alias : コマンドの別名 (エイリアス) を定義

本来は、長くて覚えにくいコマンドを別の名前で呼び出すためのものだが、以下のようにコマンドオプションを指定してデフォルトの動作を変えさせるために使うと便利である。例えばこれまでに出て来た cp, mv, rm などの危険なコマンドを

```
alias cp 'cp -i'
alias mv 'mv -i'
alias rm 'rm -i'
```

などと再定義しておけば、大切なファイルをうっかり消去してしまう可能性は減るだろう。

tcsh は起動時に~/.tcshrc または~/.cshrc を実行する (前者は後者に優先する) ので、上記のコマンドをこれらのファイルに書いておくと、ログインする時に自動的にエイリアスが定義されて便利である[21]。

定義済みのエイリアスのリストは、引数無しの

```
alias
```

で見る事ができ、エイリアス定義を削除する (例えば上記の cp の場合) には

```
unalias cp
```

とすればよい。

3.5 エディタ–mule/emacs–

ファイルの編集はエディタと呼ばれるプログラムを用いる。UNIX で利用できるエディタは多数存在するが、代表的なものは vi と emacs である。vi はどんな UNIX でも必ず使える基本的なエディタとされているが、初心者にはとっつき難い点があるので、本講義では、日本語を含む多言語で利用できるように emacs を拡張した mule(multi-lingual enhancement to GNU emacs) というエディタを標準で使う。

mule/emacs は、ファイルの種類に応じて効率的に編集を行うためのモードを持っている。モードはファイルの拡張子から自動的に選択されるので、

```
mule runge_kutta.f &
```

というように、ファイル名に拡張子を付けて起動すると便利である。図 3.1 の例では Fortran 言語モードが選択され、段付け (indent) を支援するために、Tab キーを押す事で適当な位置にカーソルがセットされるようになったりする。もちろん、ファイル名は後で指定する事もできるので、ファイル名を付けずに mule というコマンドを単独で実行してもよい。mule は後述のように、複数のファイルを同時に編集できるので、一つのファイルの編集が終了しても、mule 自身は終了せずに残しておくとよい。

mule の状態はウインドウ下部のバー上に表示される (図 3.1 参照)。各部の意味は、以下の通り。

-	言語モード (例えば日本語入力モードの時は “-A あ” となる)
E	文字コード (E:EUC、S:シフト JIS、J=JIS)
:	区切り
--又は **	編集されたバッファがファイルに保存済 (--)
	保存されていない (**)
rc_circuit.f	編集中のファイル名
(Fortran)	モード ((Fortran):Fortran モード、(C)C モード、等)
L1	カーソル位置の行番号
All	ウインドウに表示されている箇所の相対位置
	(ALL:全て表示されている、30%:ファイル先頭から 30%の位置を表示)

[21] ログイン後にこれらのファイルを実行するには、source ~/.tcshrc(または source ~/.cshrc) とすればよい。

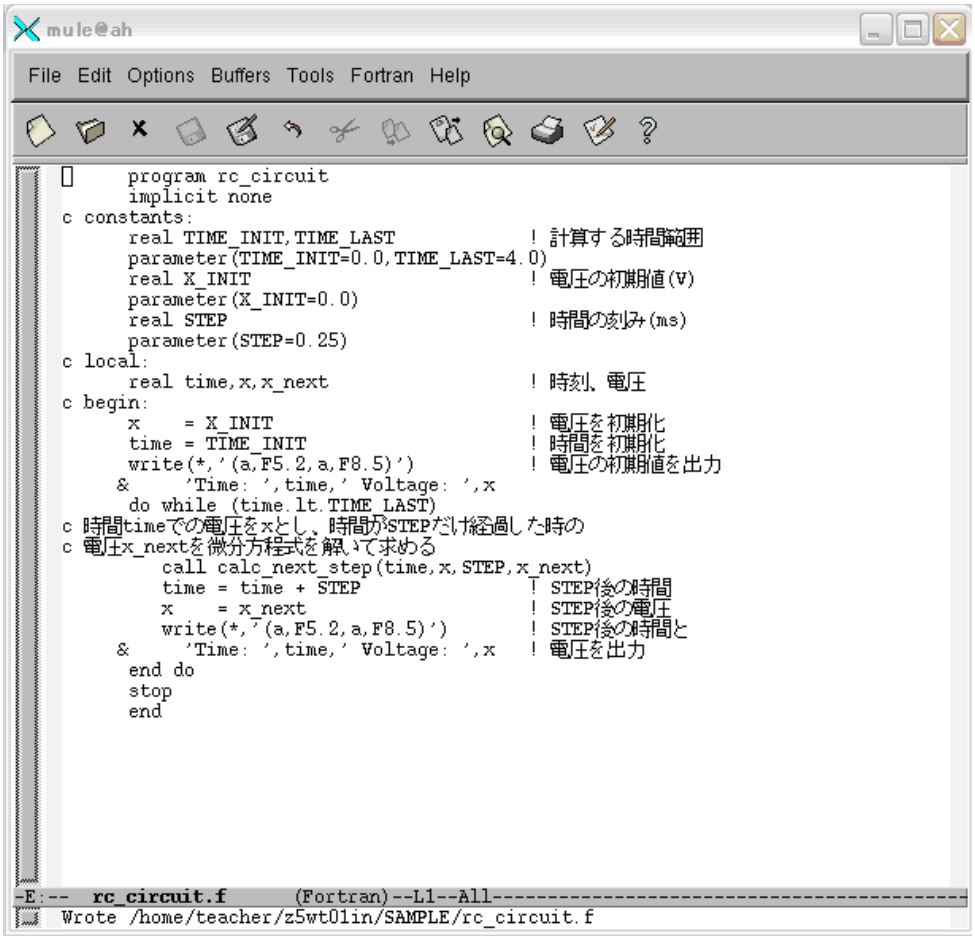


図 3.1: FORTRAN モードで起動された mule ウインドウの例。

mule/emacs では多くの操作をマウスで行う事ができ、初心者にも扱い易くなっている。文字列のカット&ペーストはもちろん。文字編集以外のほとんど全ての操作をメニューバー (図 3.1 の File Edit... と書かれた箇所) からマウス左ボタンでメニューをプルダウンして実行可能である。以下では主に、メニューバーの File メニューについて説明する。

新規ファイルの編集

mule には、新規ファイル作成の為の New File(新規作成) というメニューが無い。通常は、Buffer メニューで*scratch*というバッファを選択するか、または存在しないファイルを開く事により、新規ファイルの編集を開始する。トラブルが少ないと思われる後者の場合、File メニューの中から 1 行目の Open File... を選択する。mule ウインドウの最下行にファイル名入力を促すプロンプト (**ミニバッファ**と呼ばれる) が現われるので、適当なファイル名を入力すれば良い。

既存ファイルの編集

File メニューで Open File... と選択するところまでは、新規ファイルの編集と同じ。ファイル名入力では、tcsh と同様に、最初の数文字を入力して Tab キーを押せば、残りの文字を補完してくれる。マッチするファイルが複数ある場合は、Tab を二回押すと候補のファイル

の一覧が表示されるので、編集したいファイルへマウスポインタを移動し、中ボタンを押して選択する事もできる。

ファイルの保存

ファイルを保存するには、File メニューの Save (current buffer) を選択する。`mule` ウインドウ下側に表示されている、現在付けられているファイル名で保存される。`*scratch*`バッファを編集している (ファイル名が付けられていない) 場合でも、次の Save Buffer As... ではなく、Save (current buffer) を選択する方が無難である。なお、この場合は `mule` ウインドウ下部に新しいファイル名を問い合わせるプロンプトが出るので、適当なファイル名を入力すれば良い。

名前を付けて保存

現在付けられているファイル名と異なる名前で保存したい場合は、File メニューの Save Buffer As... を選択する。`mule` ウインドウ下部に新しいファイル名を問い合わせるプロンプトが出るので、適当なファイル名を入力すれば良い。

ファイルの挿入

カーソル位置に他のファイルを挿入するには、File メニューの Insert File... を選択する。`mule` ウインドウ下部に挿入するファイル名入力を促すプロンプトが現われる。Tab によるファイル名補完ができるのは、既存ファイルの編集で説明した通りである。

応用：類似するファイルを基に新規ファイルを作成 [22]

1. 新規ファイルを開く (File→Open File...→ 存在しないファイル名を入力)
2. ファイルを挿入 (File→Insert File...→ 基となるファイル名を入力)
3. 編集
4. ファイルを保存 (File→Save (current buffer))

バッファの切替え

File→Open File を繰り返すと、幾つものファイルを同時に編集する事ができる。それぞれのファイルの内容は、**バッファ**と呼ばれるメモリ領域にコピーされ編集される。メニューバーの Buffers をマウス左ボタンクリックすると、バッファの一覧が表示され、どのバッファを編集するか選択する事ができる。ファイル名の後に*がある場合は、ファイルに未保存である事を表し、ない場合は保存済み (ファイルの内容とバッファの内容が同一) を表す。

`mule` の終了

File メニュー最下行の Exit Emacs を選択する。最初に書いたように、一つのファイルの編集が終わったからといって、いちいち `mule` を終了させる必要はない。

これまでに説明した全ての操作は、マウスを使わなくても Ctrl キーの組合せで行う事ができる。例えば、File メニューの右側にキーの割り当てが表示されているが、その他の基本的なキー操作と併せ、表 3.1 にまとめたので参考にして欲しい。なお、C-x は Ctrl と x を同時に押し、M-x

[22] 同じ事は、既存ファイルを編集して別名で保存でも可能だが、トラブルの元 (既存ファイルを上書きしてしまう) なので初心者は避けたほうが無難である。

はAltとxを同時に押す事を意味する。操作に慣れて来たら、マウスを使わずに全てキーボード操作で済ませるようにすると、編集効率は向上し、tcsh のコマンド編集でもその多くを活用する事が出来て有益である。

3.6 プリンターへの出力

講義室には二台のプリンタが用意されており、UNIX システムから出力出来るようになっている。UNIX では、グラフックス画面描画の標準が X Window System であるのに対し、プリンタ制御には PostScript という言語が標準的に使われている。

プリンタに関する基本コマンドは以下の三つである。

```
lp -d bps1-ps ファイル名          (ファイルをプリンタへ出力)
lpstat                            (プリンタの状態表示)
cancel ジョブ番号                 (印刷をキャンセル)
```

一番目のコマンドで、bps1-ps は第二講義室の 2 台あるプリンタの内の一つを指している。他方のプリンタに出力する場合は bps2-ps とすればよい。また、第一講義室の 2 台のプリンタは aps1-ps または aps2-ps を指定すればよい。プリンタ出力コマンドを実行すると、

```
% lp -d bps1-ps laplace.ps↵
request id is bps1-ps-26 (1 file)
```

というように、ジョブ番号 (上記の場合は bps1-ps-26) が表示される。このジョブ番号は lpstat で確認する事もできる。lpstat を実行すると、

```
% lpstat↵

                        Windows LPD Server
                            Printer \\133.5.11.172\aps1-ps

Owner       Status          Jobname          Job-Id    Size   Pages  Priority
--------------------------------------------------------------------------------
```

等と表示される。Owner 欄が自分のユーザー ID のものを探せばよい (Job-ID の欄は lp コマンドを実行した時に表示されたジョブ番号になっている)。印刷を取り消したい場合は、このジョブ番号を cancel コマンドで指定すれば良い。詳しい使い方は man で各自調べる事。

なお、プリンタに出力するファイルは PostScript で書かれたファイル (通常 “PS ファイル” という) である必要がある。mule 等のエディタで編集したプログラムソース等のテキストファイルの場合は変換が必要である。具体的には、jtops というコマンドを使い、

```
jtops ファイル名 | lp -d bps1-ps
```

とすればよい。

表 3.1: mule/emacs の基本的キー操作

機能	対応キー
基本操作	
コマンドの取消	C-g
※困ったらまずはこれを実行しよう	
mule/emacs の終了	C-x C-c
ファイル操作	
現在の名前で保存	C-x C-s
名前を指定して保存	C-x C-w
ファイルの読み込み	C-x C-f
カーソル位置にファイルを挿入	C-x C-i
バッファの一覧表示	C-x b
バッファの削除	C-x k
カーソル移動	
上、下、左、右	C-p、C-n、C-b、C-f
行の先頭、末尾	C-a、C-e
次画面、前画面	C-v、M-v
バッファの先頭、末尾	M-<、M->
指定行へのカーソル移動	M-x goto-line
カーソル位置を中心に再表示	C-l
検索・置換	
前方置換、後方置換	C-s、C-r
確認しながら置換	M-%
※ y:置換する、n:置換しない、q:終了、!:一括置換	
編集	
カーソル位置の 1 文字を削除	C-d
カーソル位置から行末までを削除	C-k
※カットされた部分は後でペースト出来る	
ブロックの始点を指定	C-Space
ブロックのカット	C-w
※指定した始点からカーソル位置までカットされる	
ブロックを記憶 (カットしない)	M-w
ペースト	C-y
※マウスで選択した部分のペーストもできる	
ウインドウ操作	
バッファを上下に分割	C-x 2
バッファを左右に分割	C-x 3
今いるバッファを閉じる	C-x 0
他のバッファに移動 (バッファ分割時)	C-x o
日本語入力	
日本語モードのオン・オフ (トグル)	C-\

3.7 プログラミング言語

計算機のプログラミング言語には、大きく分けて**コンパイラ**言語と**インタープリタ**(または**スクリプト**) 言語がある。人間にとって理解が容易なプログラムを、計算機が直接実行可能な形式 (バイナリ) に変換する作業を**コンパイル**と言い、コンパイル結果を一旦ファイルに出力してからそれを実行させるのがコンパイラ言語である。このタイプの言語は、計算機が直接解釈するので実行が速い点で有利であるが、プログラムを修正する度にコンパイルが必要となるためデバッグ作業には不利である。これに対しインタープリタ言語では、プログラムを逐次的に (一行一行) 解釈・実行させるため実行速度が遅いという不利はあるが、修正を加えても即実行可能なのでデバックが容易である。

3.7.1 数値計算用コンパイラ言語–FORTRAN 77, C, Fortran 90

数値計算は実行速度が重要であるので、通常コンパイラ言語が用いられる。本講義ではFORTRAN 77 を使用する。C に精通している者は C を使っても構わないが、物理学の分野では FORTRAN 77 で書かれた資産が多いため、FORTRAN 77 のプログラムを読んで理解したり、利用したりといった最低限の事は出来るようにする事。

FORTRAN 77 の新しい言語規格に Fortran 90 がある。FORNTAN 77 は多くの UNIX システム上で利用可能であるが Fortran 90 は普及途上である事、多くの資産が FORTRAN 77 で書かれている事から、本講義では Fortran 90 については特に説明しない。しかしながら、Fortran 90 の言語規格を用いると、FORTRAN 77 に比べスマートなプログラムが書けるので、実力のある人は挑戦してみて欲しい。

前節でも述べた通り、ファイルは拡張子によってその属性が端的に表されており、コンパイラ言語関係では一般に以下のような約束がある。

`.f`	FORTRAN 77 言語ソース
`.f90`	Fortran 90 言語ソース
`.c`	C 言語ソース
`.o`	オブジェクトファイル
`.a`	ライブラリ (UNIX では archive と呼ばれる)

ただし実行可能ファイルには (`.exe` を付ける Windows 等と異なり) 拡張子を付けないのが一般的である。また他の多くの OS では

ソース → [コンパイラ] → オブジェクト (ライブラリ) → [リンカ] → 実行可能ファイル

という手順を踏むのが一般的であるが、UNIX の場合はリンカ (`ld` というプログラム) は必要に応じてコンパイラから自動的に呼び出されるため、多くの場合**一つのコマンドでソースから実行ファイルが生成**される。具体的には、言語に応じて以下のコマンドを実行すればよい。

`frt samle1.f sample2.f ··· -o sample` (FORTRAN 77)
`frt samle1.f90 sample2.f90 ··· -o sample` (Fortran 90)
`cc samle1.c sample2.c ··· -o sample -lm` (C)

ソースファイルの個数は一つでも複数でも良く、複数指定した場合は、全てをコンパイルした後に `ld` がリンクしてくれる。`-o` オプションは実行可能ファイルの指定を表しており、上記の例で

は sample という実行ファイルが生成される (-o と sample の間は空白があってもなくてもよい)。-o オプションで出力ファイルが指定されなかった場合は、a.out というファイルが生成される約束になっている[23]

> cc(C compiler) のみ-lm というオプションが付けられているが、-l はライブラリの指定を表し、m は数学関数ライブラリを指定している (-l と m の間にスペースを入れてはいけない)。数学関数 (sin, cos, exp 等) は C 言語の仕様に含まれていないのでこのようなオプションが必要である (ソースの中にも#include <math.h>が必要)。

3.7.2 拡張版 FORTRAN 77

FORTRAN 77 の優位性は既に述べた通りだが、純粋な規格通りの FORTRAN 77 は流石に古めかしく使い難い。今日利用可能な FORTRAN 77 コンパイラはいずれも様々な拡張機能を備えているが、その中で共通的に使えるものを以下にリストする[24]。本講義 (サンプル) でも、分かり易さのため積極的に使用している。

- アルファベット小文字が使える (FORTRAN 77 規格では全て大文字) [25]
- 長い変数名やサブルーチン名が使える (FORTRAN 77 規格では最大 6 文字)
- !以降行末までをコメントとして無視する
- implicit none(変数を宣言無しに使用する事を禁止) が使える
- 変数宣言文のなかで/···/により初期値を指定できる
- do ··· end do や do while ··· end do が使える (行番号が要らない)
- include 文によるファイル読み込みが可能

3.7.3 スクリプト言語

シェルは元来対話的にコマンドを入力・実行するものだが、長いコマンド列を何度もタイプし直す代わりに、あらかじめファイルに書いておいて一気に実行させる事もできる。このファイルは台本 (script) に準えて**スクリプト**と呼ばれるが、派生的に、ファイルのコマンド列を逐次的に解釈・実行する言語は一般に**スクリプト言語**と呼ばれる。古くからあるスクリプト言語の代表は sed と awk であるが、最近は perl に始まり tcl/tk, python, ruby など枚挙に暇がない。これらをいちいち紹介することは不可能なので、興味のある人は適当な解説書なりインターネット上の解説を参照して欲しい。

[23] a.out は assembler output の意味で、UNIX 黎明期の歴史を引きずる用語である。
[24] 大部分は、MIL-STD 1753 という米軍の FORTRAN 拡張規格に含まれているものである。
[25] ただし、大文字と小文字は区別しない。

C 言語を信奉する人の中には「全てを C で書こう」とする人がいるが、多くの便利なスクリプト言語が整備された現在では、必ずしも賢い選択では無い。スクリプト言語の多くは、既存のコマンドや C 言語とのインターフェイスを備えているので、実行効率 (速度) を重視する部分は C 言語等のコンパイラ言語を用い、ユーザーインターフェイスやファイル操作などの速度は必要としないが記述の煩わしい部分はスクリプト言語を用いて簡便に記述するという賢いプログラムスタイルが可能となっている。

計算・メモ用余白