UNIX/FORTRAN/ Bourne Shell Script

自習テキスト

花崎　直太

新田　友子

はじめに

このテキストは水文学を学ぶ学部生、大学院生を対象とした計算機の自習教材です。このテキストを通読すると、読者は水文学のデータ解析やコンピュータシミュレーションを行う際に必要となる UNIX、FORTRAN、Bourne Shell Script に関する基本的な技術を身につけることができます。

このテキストは東京大学生産技術研究所沖研究室で行われていた初心者向けの UNIX 講座の講義録が元になっています。沖研究室は地球水循環システムの研究を行う研究室で、地球水循環のコンピュータシミュレーションやデータ解析で大きな成果を挙げています。この研究室で修士論文や博士論文を書くには、学生は UNIX とプログラミングに精通することが不可欠です。

さて、沖研究室に在籍する学生はほぼ全員、夏学期に沖教授が担当する「Advanced Hydrology」（講義の日本語名はありませんが無理に訳せば高等水文学）という講義に出席します。2001 年頃、この講義には次のような宿題がありました。

> Read the following article:
> *Xie, P., and P. A. Arkin (1997), Global Precipitation: A 17-Year Monthly Analysis Based on Gauge Observations, Satellite Estimates, and Numerical Model Outputs, Bull. Amer. Meteor. Soc., 78, 2539-2558.*
> - Summarize their methodology
> - List up advantage/disadvantage
> - Compare the data with other data
>
> (e.g. station observation, global dataset, etc.)
> - See the site below for original data
>
> ftp://ftp.cpc.ncep.noaa.gov/precip/cmap/monthly/

要約すると、全球の月単位の降水量データ CMAP (CPC Merged Analysis of Precipitation) の作成を報告した Xie and Arkin (1997)を読み、手法や特徴をまとめた上で、実際に彼らのデータを入手して他の降水量データと比較せよ、ということになります。

受講者（多くは沖研究室に入ったばかりの修士１年生、博士１年生）は、宿題の後半に苦労しました。CMAP は全球を 2.5 度×2.5 度の解像度でカバーする格子データです。地球は経度方向に 360 度、緯度方向に 180 度ありますから、ひと月のデータは 144x72=10368 地点の数値からなります。このことは Microsoft Excel のスプレッドシートに、144 列 72 行にわたって数値が並んでいることを想像するとよく分かると思います。ウェブサイトからダウンロードできるのは、こうした数値が、10368 地点、１２カ月分縦に並んだテキストファイルで、10368x12=124416 行あります。受講者の多くは Microsoft Excel を使うことはできましたが、当時の Excel は 65536 行のテキストファイルまでしか開くことができず、まずどうやってファイルを開くか、というところから悩まされたのです。幸いにファイルが開けたとしても、次はどうやって図を作るかに苦労しました。CMAP は全球をカバーするデータですから、降水量の地図を描きたくなるものですが、当時の東京大学社会基盤工学科には GIS ソフトウエアの体系だった講義がなく、多くの

学生には地図の描き方が分かりませんでした。結論から言うと、沖研究室にあった UNIX コンピュータを使いこなさない限り、沖教授を満足させるような宿題は提出できなかったのでした。

ここまでの話をまとめると、沖研究室の大学院学生は UNIX に精通する必要があり、また夏学期に行われる沖教授の講義の宿題を提出するために UNIX を利用した CMAP データの解析を早急に習得する必要がありました。後者で必要な技術を挙げると次のようになります。

- UNIX の操作
- FORTRAN[1]を使ったプログラミング
- Generic Mapping Tools (GMT)[2]ソフトウエアを使った地図の作成
- Bourne Shell Script を使った大量のデータを効率的に処理するための技術

当時大学院学生だった筆者は 2002 年～2005 年度の UNIX 講座を担当しており、こうしたニーズに応えるため講義を編成し、講義録を作成しました。その後、多くの人の協力を得て文章や内容を充実させ、講義録をこのテキストに再編しました。このテキストで解説する技術は宿題の提出だけでなく、修士論文や博士論文の研究にもきっと役立つはずです。また、全球水資源モデル H08[3]に関心がある人にも役立つはずです。H08 で使われている技術の基礎は全てこのテキストから学ぶことができるでしょう。

このテキストは「初学者でも飽きずに最後まで続けられること」を優先して構成されています。特に 1 章読み進めるごとに読者が宿題提出への手ごたえを感じられるように配慮したつもりです。ただし、その結果として、説明の順番が前後したり、体系的な説明になっていなかったりする個所があります。また、このテキストは UNIX, FORTRAN, Bourne シェルスクリプトを網羅的に解説することを意図したものではなく、むしろ、市販の教科書には詳しく書かれないことが多いけれども、水文学のデータを扱う時に必須の技術に絞って解説したものといえます。Appendix D に参考文献を挙げましたので、このテキストと並行して読み進めてほしいと思います。

筆者自身の経験から言うと、UNIX を知っているか知らないかで、またプログラミングできるかできないかで、扱えるデータやシミュレーションが全く変わります。このテキストが読者の UNIX やプログラミングに取り組むきっかけになることを、筆者は願ってやみません。

謝辞：第３版の編集にあたり、上野博史さんの助力を得ました。ここに記して感謝します。

[1] FORTRAN は地球科学においては今でも非常によく使われている言語です。大気海洋結合モデルをはじめとする地球科学の重要なコンピュータプログラムの多くが FORTRAN で記述されています。
[2] http://gmt.soest.hawaii.edu/
[3] Hanasaki et al. 2008a,b (http://direct.sref.org/1607-7938/hess/2008-12-1007, http://direct.sref.org/1607-7938/hess/2008-12-1027)で発表された全球水資源モデルのこと。地球の自然の水循環と主要な人間の水利用を同時に計算することのできる水文モデルである。

目次

第 1 章

UNIX の基本操作(1) 基本的な UNIX コマンド

どうすれば UNIX 環境を構築できるのでしょうか？どうやって UNIX を操作するのでしょうか？第1章では UNIX の基本操作について説明します。

1.1 UNIX 環境の構築

このテキストを利用するには UNIX 環境の構築が必要です。構築の仕方は付録（Appendix）に書かれています。図 1-1 のフローチャートの質問に答えると、あなたに適した UNIX 環境を構築するため、どの Appendix を読むべきかが分かるはずです。

研究室に UNIX サーバがある場合は、Appendix A を実施した後、Appendix D を実施して下さい。研究室に UNIX サーバがなく、新しい計算機を買う経済的余裕がある場合は、Appendix B を実施した後、Appendix D を実施して下さい。研究室にサーバがなく、新しい計算機を買う経済的余裕もない場合は、Appendix C を実施した後、Appendix D を実施して下さい。

- Appendix A：Windows 機から UNIX サーバにログインする
- Appendix B：Mac を UNIX 環境として利用する
- Appendix C：ubuntu とソフトウエアの導入
- Appendix D：テキスト教材の導入と環境設定

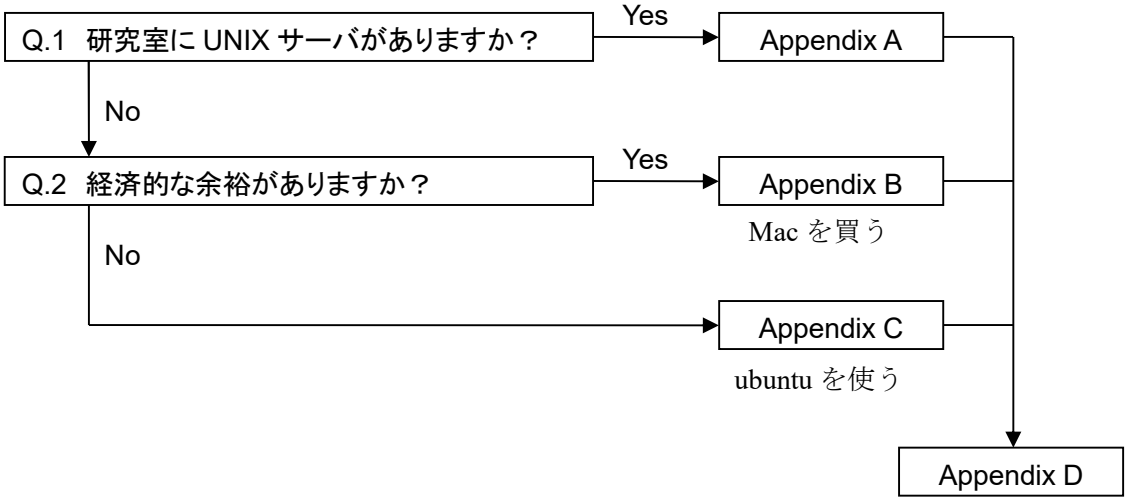


図 1-1 UNIX 環境構築のフローチャート

1.2 UNIX コマンド①――ファイルの管理

Windows ではマウスを使ってアイコンをクリックしてコンピュータを操作します。このような概念を GUI (Graphic User Interface)と言います。これに対して、UNIX ではキーボードを使って端末にコマンドを打ち込んでコンピュータを操作するのが基本です。このような概念を CUI (Command User Interface)と言います。UNIX 操作の基本となるコマンドを

表 1-1 から表 1-4 にまとめました。一つずつ端末に打ち込んで機能を確認してみてください。

表 1-1 はファイル管理のためのコマンドです。Windows の Explorer での操作に相当します。以下にディレクトリという言葉があらわれますが、Windows のフォルダと同じ意味です。表の中にある%はプロンプト（ターミナルに表示される「pc090-041[35]」などの文字列のこと）なので、入力しないで下さい。

表 1-1 ファイルの管理（***directory*** は任意のディレクトリを、***file*** は任意のファイルを示す）

今いるディレクトリ名を表示する(pwd は present working directory の略)	`% pwd`
今いるディレクトリにあるファイル名・ディレクトリ名を表示する(ls は list の略)	`% ls`
より詳細なファイル、ディレクトリ情報を表示する(l は long の略)[4]表示される内容は左から順に、アクセス権限，リンク数，所有者，グループ所有者，ファイルサイズ，月，日，年または時刻，ファイル名 or ディレクトリ名	`% ls -l`
ディレクトリを移動する(cd は change directory の略)[5]	`% cd` ***`directory`***
ひとつ上のディレクトリに移動する（ピリオド2つは一つ上のディレクトリを示す）	`% cd ..`
ホームディレクトリ[6]に移動する（チルダはホームディレクトリを示す）	`% cd ~`
今いるディレクトリに移動する（ピリオド1つは今いるディレクトリを示す）	`% cd .`
ファイルの内容を表示する	`% less` ***`file`*** `% more` ***`file`*** (1行スクロールするには enter を、 1画面スクロールするには space を、 終了するには q と入力する) `% head` ***`file`*** （ファイルの先頭の 10 行だけ表示する） `% tail` ***`file`*** （ファイルの末尾の 10 行だけ表示する）

[4] この例のように、コマンド（ls）の後に続くハイフン（-）から始まる文字列をオプションと言います。コマンドにオプションを付けることにより、コマンドの機能を変えることができます。
[5] この例のように、コマンド（cd）の後に続く文字列を引数（argument）と言います。引数を与えることにより、コマンドに情報を与えることができます。
[6] ホームディレクトリ(~)とはユーザが最初に所有するディレクトリのこと。Windows でいうところの C:\Documents and Settings\guest。Mac OS X の場合、ログイン名 guest のホームディレクトリは/~であるが、ほとんどの UNIX 系 OS では/home/guest である。

ファイルを編集する（新しくウィンドウを開く）	% emacs *file*
ファイルを編集する（ウィンドウを開かない）	% emacs –nw *file*
ファイルをコピーする（cp は copy の略）	% cp *file1 file2* （コピー元は file1、コピー先は file2）
ファイルを移動する／名前を変更する（mv は move の略）	% mv ***file1 file2*** （移動元は file1、移動先は file2）
ファイルを削除する（rm は remove の略）	% rm ***file***
新しいディレクトリを作成する（mkdir は make directory の略）	% mkdir ***directory***
空のディレクトリを削除する（rmdir は remove directory の略）	% rmdir ***directory***
空でないディレクトリを削除する（-r は recursive の略で、再帰的にという意味）	% rm –r ***directory***
ディレクトリをコピーする	% cp –r ***directory1 directory2***
パーミッションを変更する（chmod は change mode の略）※1.7 参照	% chmod ***abc file*** a:ユーザの読取権限(+4), 書込権限(+2), 実行権限(+1) b:グループの読取権限(+4), 書込権限(+2), 実行権限(+1) c:その他の読取権限(+4), 書込権限(+2), 実行権限(+1)
パーミッションを変更する（もう別の書式）※1.7 参照	% chmod ***abc file*** a:ユーザ(u),グループ(g),その他(o) b:加える(+), 外す(-) c:読取権限(r), 書込権限(w), 実行権限(x)
ファイルを圧縮する（gzip は GNU zip の略）	% gzip ***file***
ファイルを解凍する（gunzip は GNU unzip の略）	% gunzip ***file***
終了する	% exit

1.3 UNIX コマンド②――ファイルの編集

Windows のメモ帳のようにテキストファイルを編集するには Emacs というソフトウエアを使います。Emacs はほとんどの UNIX コンピュータにインストールされているはずです。Emacs はキーボードショートカットを使ってファイルを保存したり、文章をコピーしたりします。表 1-2 には emacs の編集に関するコマンドをまとめました。

表 1-2 emacs を使ったファイルの編集。Ctrl-x とは Ctrl キーと x を同時に押すことを示す。また、Ctrl-x + Ctrl-s は Ctrl キーと x を同時に押したあと、Ctrl キーと s を同時に押すことを示す。

Emacs の起動	`% emacs` ***file***
保存	Ctrl-x + Ctrl-s
終了	Ctrl-x + Ctrl-c
コマンドのキャンセル （コマンドを打っていてうまく動作 しなくなった場合、この操作を2～3回 繰り返すと元に戻ることが多い）	Ctrl-g
文字を削除する（delete）	Ctrl-d
行を削除する (kill)	Ctrl-k
切り取り[7]	Ctrl-Space (始点を指定) Ctrl-w（終点を指定）
貼り付け (yank)	Ctrl-y
カーソルを右に移動 (forward)	Ctrl-f
カーソルを左に移動 (backward)	Ctrl-b
カーソルを上に移動 (previous)	Ctrl-p
カーソルを下に移動 (next)	Ctrl-n
画面を下へスクロールする	Ctrl-v
最後の画面へスクロールする	Esc + >
画面を上へスクロールする	Esc + v
最初の画面にスクロールする	Esc + <
検索	Ctrl-s
置換	Esc + %
一括置換	Esc-x replace string

1.4 UNIX コマンド③——その他のコマンド

表 1-3 には画像ファイルの表示や印刷など、よく使うコマンドをまとめました。

表 1-3 よく使うコマンド

Firefox ブラウザ[8]の起動	`% firefox`
JPEG や GIF ファイルの表示[9]	`% display` ***file***
画像ファイルの変換[10]	`% convert` ***file1 file2***
印刷[11]	`% lpr -P`***printer file***

[7] Windows のようにコピーがないので、コピーするときは一度カットしてからもう一度ペーストする。
[8] コンピュータに Firefox がインストールされていない場合はその他のブラウザ名を打ち込んでみてください。
[9] Imagemagick というソフトウエアがインストールされていないと使えません。詳しくは Appendix C を見てください。
[10] 同上。

印刷ジョブの表示	% lpq -P***printer***
印刷ジョブの削除	% lprm -P***printer jobID***
コンピュータにログインしている ユーザの一覧を表示する	% who
日付を表示する	% date
year 年のカレンダーを表示する	% cal ***year***
ファイルの行、単語数、文字数を表示する	% wc ***file***

1.5 UNIX コマンド④——困ったときは

困ったときには、表 1-4 のコマンドを試してみましょう。

表 1-4 困ったときのコマンド

コマンドのマニュアルを表示する	% man ***command***
プロンプトが返らなくなってコマンドを中断する	Ctrl-c
コマンドの状態と process ID を表示する	% top
コマンドを強制終了させる	% kill ***processID***
コマンドを強制終了させる（強力）	% kill -9 ***processID***

1.6 パーミッション

パーミッションは UNIX で非常に重要な概念なので必ず覚えましょう。まず、ユーザのファイルやディレクトリへのアクセスは３通りあります。

- 読取：ファイルを開いて読む
- 書込：ファイルを新しく作る。または既存のファイルの内容を改変する。
- 実行：ファイルにコマンドが書かれているとみなして実行する。

次に UNIX では全てのファイルとディレクトリは誰か一人のユーザによって所有されています。またユーザが集まってグループを作ることができます。よってファイルやディレクトリとユーザの関係も３通りあります。

- User: ファイル・ディレクトリの所有者であるユーザ
- Group: 所有者ではないが所有者と同じグループに所属するユーザ(group)
- Others: 所有者でなく、かつ所有者と同じグループに所属していないユーザ

おそらく、ファイルの実行というのが分かりにくいと思うので、実際に試してみましょう。emacs を使って次のようなテキストファイルを作成しましょう。

```
pwd
ls
who
```

[11] 例えば、flood というプリンタで test.txt というファイルを印刷する場合は、
「%lpr –Pflood test.txt 」としましょう。

これを“example1.txt”という名前で保存してください。これは Windows で普通のテキストファイルを作るのと同じですね。

次にパーミッションを変更して、実行権限を付け加えましょう。このファイルは自分で読み、編集し、実行するものですね。ただし、同じグループのユーザや他人には読まれてもよいけれども、編集されたり、実行されたりしたくないとします。その時は、

```
% chmod 744 example1.txt
```

とします[12]。あるいは

```
% chmod u+x example1.txt
```

としても同じです[13]。こうすると、あなたはこのファイルが実行可能になります。それでは、example1.txt を実行しましょう。

```
% example1.txt
```

UNIX では Windows と違って、実行権限さえあれば、テキストファイルも実行できることが分かりましたね。この場合、テキストファイルに書かれた内容が、コマンドとして解釈されます。

宿題

1. 現在のディレクトリを確認しましょう。
2. あなたのホームディレクトリに移動しましょう。(~)
3. “CMAP”という名前のディレクトリを作成しましょう。
4. 作成したディレクトリ“CMAP”に移動しましょう。
5. ~/unix_semi/lesson1 から
   - global.sh
   - grad.cpt
   - data.xyz をコピーしましょう。[14]
6. global.sh のパーミッションを変更して、実行権限を付け加えましょう。
7. global.sh を実行しましょう。
8. 出力された画像ファイル”image.eps”を表示しましょう。これは次章以降で説明しますが、1979 年 1 月の世界の降水量のデータを示しています。
9. 画像ファイル”image.eps”を EPS 形式から GIF 形式に変換しましょう。

※宿題の解答例は~/unix_semi/lesson1/exercise1.txt にあります。

[12] User は読み、書き、実行できるのだから、表 1-1 を参考にすると、4+2+1=7 となる。Group と Others は読めるだけなのだから、4+0+0=4 となる。これをつなげると 744 となる。
[13] User に実行権限を足すのだから、u+x となる。これ以外は変更なし。
[14] 見つからない場合は、もう一度 Appendix D を実施してみてください。

第 2 章

UNIX の基本操作(2)　Redirect/Pipe/Wildcard/Awk

第1章で使ったファイル global.sh を開いてください。このファイルは多数の短いコマンドから成り立っています。注意深く見ると、＞、＞＞、＜、＜＜、｜のような特殊な文字がたくさん使われていることがわかります。これは UNIX を操作する上で大変重要な記号です。第2章では、これらの記号について勉強しましょう。

2.1　Redirect

「＜」と「＞」の記号はリダイレクト(redirect)と呼ばれます。リダイレクトはコマンドの出力をファイルに書き込みたい場合や、コマンドへの入力をファイルから読み込みたい場合に用います。リダイレクトの種類を表 2-1 にまとめました。

表 2-1　リダイレクトの種類

コマンドの出力をファイルに書き込む もとのファイルは上書きされてなくなる[15]	% ***command*** > ***file***
コマンドの出力をファイルの末尾に書き足す	% ***command*** >> ***file***
ファイルを入力としてコマンドを実行する	% ***command*** < ***file***
次に「EOF」という文字列が入力されるまで キーボード入力された文字列を 入力としてコマンドを実行する	% ***command*** << EOF *strings* EOF

【例１】

ターミナルに次のように打ち込んでみましょう。date は現在の日付や時刻を出力するコマンド、wc は文字数や行数を数えるコマンドで、行の数、単語の数、文字の数の３つが出力されます。

```
% date > redirect.txt
% more redirect.txt
% date >> redirect.txt
% more redirect.txt
% date > redirect.txt
% more redirect.txt
% wc < redirect.txt
```

エラーが出るときは
% date >! redirect.txt
を試しましょう

[15] ターミナルに%echo $SHELL と打ってみて tcsh か csh と表示される場合はリダイレクトによる上書き防止機能が有効になっている。既にある redirect.txt を上書きする場合は、% command >! file とする。

```
% wc << EOF
Hello.
My name is Naota.
EOF
```

```
% wc << EOF >> redirect.txt
Hello.
My name is Naota.
EOF
% more redirect.txt
```

【宿題１】

1. 今いるサーバにログインしている人の一覧を書き込んだファイルを作りましょう。（表 1-3 にあるコマンド 「who」 をうまく使いましょう）
2. 2004 年から 2005 年までのカレンダーを書き込んだファイルを作りましょう。（表 1-3 にあるコマンド 「cal」 をうまく使いましょう）

※宿題１の解答例は~/unix_semi/lesson2/exercise1.sh にあります。

2.2 Pipe

「｜」の記号はパイプ(pipe)と呼ばれます。パイプはコマンドの出力を次のコマンドへの入力として受け渡します（表 2-2）。

表 2-2 Pipe

まずコマンド1が実行され、 次にコマンド1の出力を入力として コマンド2が実行される	% ***command1*** ｜ ***command2***

【例２】

```
% cd ~/unix_semi/dat_bin
% ls -l
% ls -l | more
% ls -l | tail
% ls -l | head
```

【宿題２】

1. あなたのホームディレクトリにあるファイルとディレクトリの数を数えましょう。（コマンド「wc」をうまく使いましょう）
   - redirect を使ってやってみましょう
   - pipe を使ってやってみましょう

※宿題２の解答は~ /unix_semi/lesson2/exercise2.sh にあります。

2.3 Wildcard

「?」と「*」の記号はワイルドカード(Wildcard)と呼ばれ、それぞれ任意の 1 文字や文字列を表します（表 2-3）。

表 2-3 wildcard

Null[16]を含む任意の文字列	*
任意の1文字	?

【例3】

```
% cp -r ~/unix_semi/lesson2/wildcard .
% cd wildcard
% ls -l
% ls -l *txt
% ls -l test*
% ls -l te?t1.txt
```

ピリオドを忘れないで

t の後は数字

【宿題３】

1. 例３でコピーしたディレクトリ「wildcard」に移動しましょう。
2. ファイル名に「1（数字)」を含むファイルを削除しましょう。
3. ディレクトリ wildcard の中のすべてのファイルを削除しましょう
4. 例３のように「–r」オプションを使わずに、~/unix_semi/lesson2/wildcard からすべてのファイルをコピーしましょう。(「*」 をうまく使いましょう)

※宿題３の解答例は~/unix_semi/lesson2/exercise3.sh にあります。

2.4 AWK

AWK は一種のプログラミング言語で、主にテキストを処理するために使われます。コンパイルする必要がないインタプリタ言語で、直接ターミナルに打ち込んで使えます。文法は C 言語とよく似ています。AWK を使えばかなり高度なプログラミングも可能ですが、ここでは表 2-4 の２つの使い方だけ覚えましょう。表 2-4 で「‘」と「’」で囲まれた部

[16] ゼロ文字でもよいということ

分が AWK の解釈するソースコードです。

表 2-4 AWK のコマンド

ファイルの N_1, N_2 列目のデータを書き出す	`% awk '{print $N1, $N2}' file`

M 列目が m の時、N 列目のデータを書き出す[17]
例）1 列目が"a"のとき、2列目のデータを書き出す。
例）1 列目が 12 のとき、2列目のデータを書き出す。
	`% awk '($M==m){print $N}' file`
	`% awk '($1=="a"){print $2}' file`
	`% awk '($1==12){print $2}' file`

【例４】

```
% cd ~
% ls -l > home.txt
% awk '{print $9}' home.txt
% awk '($5==512){print $9}' home.txt
```

さて第 1 章で使ったファイル「data.xyz」をもう一度開いてみてください。これは CMAP の 1979 年 1 月の降水量データを 1 列目に経度、2 列目に緯度、3 列目に降水量（単位は mm/day）の順で記録したデータです。CMAP は 2.5 度×2.5 度で全球をカバーするので、全部で 144x72=10368 行あります。次の第 3 章で解説しますが、global.sh は data.xyz を読み込んで描画を行います。つまり、第 1 章で作成した図は data.xyz を可視化したものだったのです。

次に CMAP のウェブサイト（ftp://ftp.cpc.ncep.noaa.gov/precip/cmap/monthly/）で配布されているファイルを見てみましょう。まず、1979 年の月単位のデータファイルをダウンロードしましょう[18]（※Appendix B 参照）。このファイルは圧縮されているので解凍しましょう。gz 圧縮ファイルの解凍には gunzip コマンドを使います。

```
% gunzip cmap_mon_vXXXX_79.txt.gz
```

[17] m が文字列の時、「“」と「”」で囲みます。m が実数か整数の時はそのまま書きます。

[18] 最も簡単なのは Firefox などの web ブラウザを使ってダウンロードする方法ですが、コマンドを使ってダウンロードする場合は、次のようにしましょう。

```
% ftp -i ftp.cpc.ncep.noaa.gov      ←FTP サーバにログインします。※1
% ls                                 ←リストを表示します。
% cd precip/cmap/monthly            ←ディレクトリを移動します。
% binary                            ←ファイル転送モードを binary にします。※2
% get cmap_mon_vXXXX_79.txt.gz      ←ファイルをダウンロードする。※3
% quit                              ←FTP を終了します。
```

※1：途中で Name を聞かれたら、anonymous と打ち込みましょう。次に Password を聞かれたら、自分の Email アドレスを正確に入力しましょう。ログインできるはずです。ただし、ftp は 2022 年現在ではかなり古くなっており、環境によってはコマンドが動作しないかもしれません。以下のサイトを参照してください。https://ftp.cpc.ncep.noaa.gov/（最終アクセス 2022/06/20）。

※2：ファイル転送モードには binary と ascii があります。binary はファイルをそのまま転送します。ascii は OS による改行コードの違いを自動的に変換して転送します。テキストファイルを転送する時以外は必ず binary モードを選択しましょう。

※3：% get file がファイルをダウンロードするコマンドです。この章でやったように「*」などの wildcard を使ったファイル指定をする場合は、% mget file を使います。例えばディレクトリにある全てのファイルをダウンロードしたい場合は、% mget *とします。

さて、ファイルの内容を表示させましょう。図 2-1 に凡例を示しますが、data.xyz とは書式がかなり異なることが分かるでしょう。

```
% more cmap_mon_v0705_79.txt
```

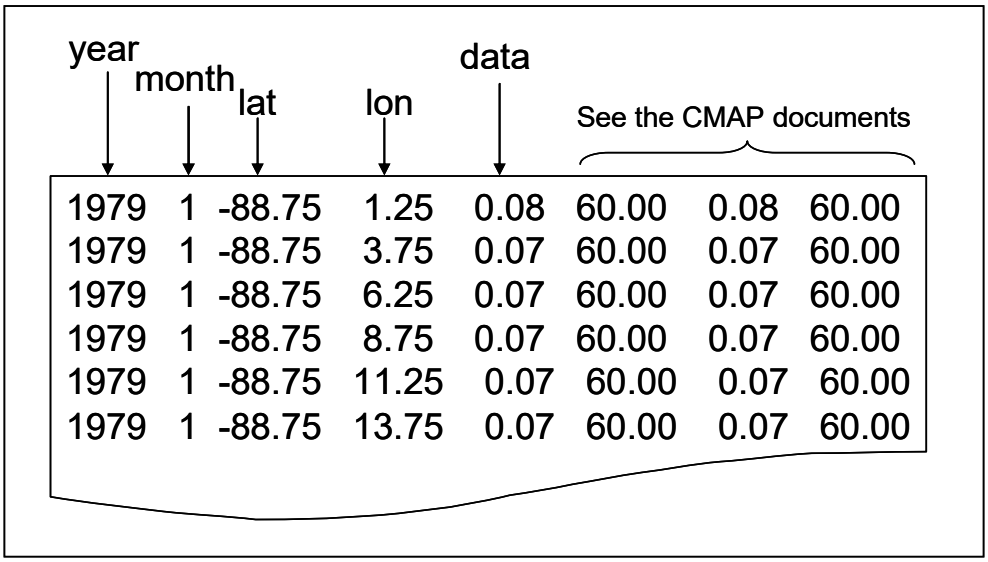


図 2-1 CMAP ファイルの凡例

【宿題４】

1. 図 2-1 にあるのがあなたのダウンロードした cmap_mon_v0705_79.txt の凡例です。これを参考にして、AWK を使って data.xyz と全く同じ形式のファイルを作りましょう。ここで注意することは、data.xyz は 1979 年 1 月のデータを経度、緯度、降水量という書式で書き出したファイルです。

   ※ 宿題４の解答例は~/unix_semi/lesson2/exercise4.sh にあります。
   ※ 出力の桁数を揃えたい場合は、インターネット検索してください。

第 3 章

Bourne シェルスクリプト(1)　GMT コマンドを組み合わせる

第1章では、global.sh というファイルを実行して地図を描きましたが、この地図はどのようにして描かれたのでしょうか。どのようにすれば、編集できるのでしょうか。

3.1　GMT とは

第 1 章の復習をしましょう。以下のようなコマンドを打ち込み、図 3-1 のような地図を出力したのでしたね。

```
% global.sh
% display image.eps
```

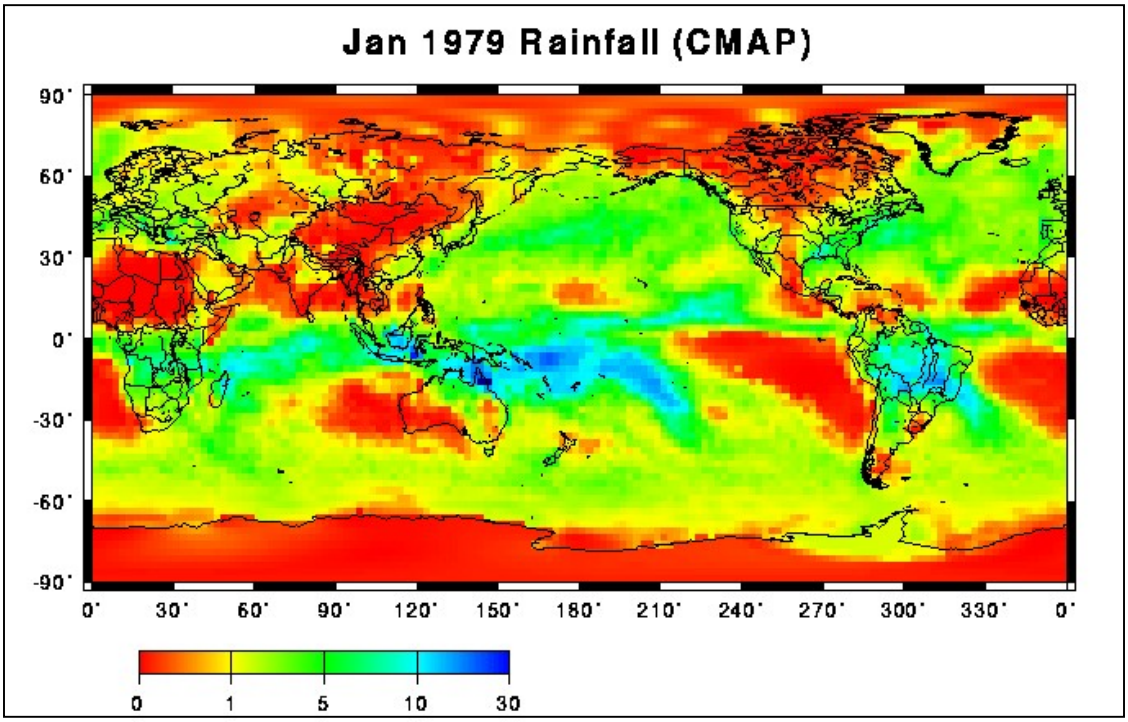


図 3-1　global.sh で作成された図

この作図には Generic Mapping Tool (GMT)[19]というフリーソフトウエアが使われていました。GMT は ls や cd と同じような形式の多数のコマンドからなります。global.sh ではこれらの GMT コマンドを組み合わせることにより、全球の降水量の地図を描いていたのです。

GMT を利用して作図するには以下の３つのファイルが必要です。

- ソースファイル（global.sh）
- データファイル（data.xyz）

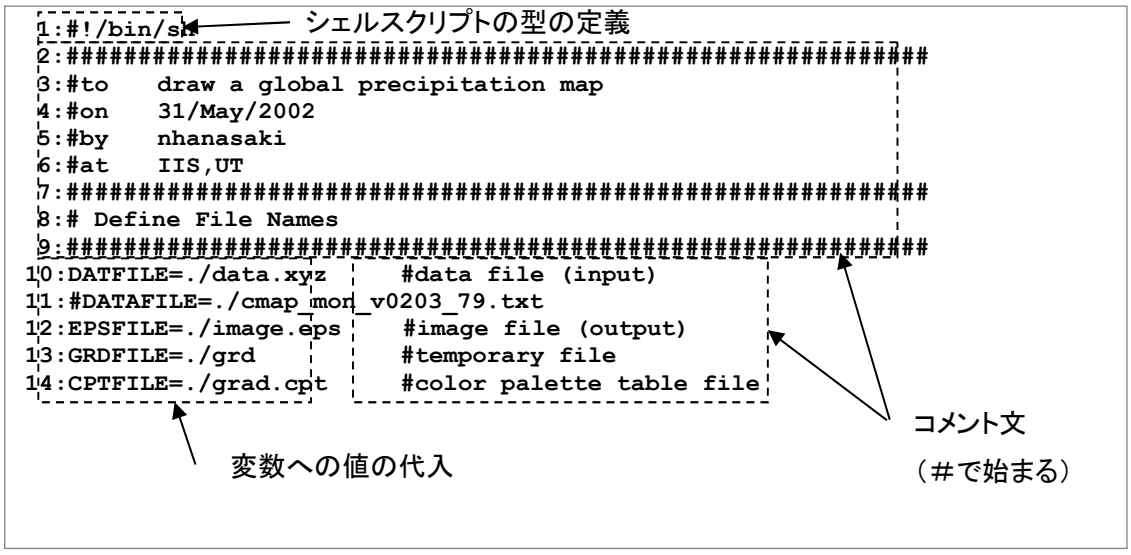
[19] http://gmt.soest.hawaii.edu/

- CPT ファイル（grad.cpt）

それぞれのファイルの役割を解説していきます。

3.2　GMT のシェルスクリプト

global.sh をテキストエディタで開いてみましょう。1.7 で pwd, ls, who とコマンドを書いたテキストファイルに実行権限を加えて実行したことを覚えているでしょう。この global.sh も同じで、たくさんのコマンドが書きこまれています。ただし、global.sh は Bourne Shell という文法に従って記述されており、それによってコマンドの実行が制御できるようになっています。こうしたファイルを Bourne シェルスクリプトと言います。つまり、global.sh は GMT コマンドを含む Bourne シェルスクリプトである、ということができます。以降では、ファイルを３つのブロックに区切って、解説していきます。それぞれ第１ブロック、第２ブロック、第３ブロックと呼ぶことにしましょう。



第１ブロックについて説明しましょう。１行目に「#!/bin/sh」と書いてありますが、これはこのファイルが Bourne シェルスクリプトであることを示す文字列です。２行目から９行目まで「＃」で始まる行が続きますが、「＃」以降の文字列はコンピュータに解釈されません。よって、この例のように、覚え書きや区切りをいれるために使われます。こうした文をコメント文と言います。

１０行目から１４行目では、２つの文字列が「=」で左右につながっていますね。「=」の左側の文字列は「変数」です。空の箱だと考えてください。「=」の右側の文字列は「値」です。何かの物だと考えてください。「変数」と「値」を「=」で結ぶことにより、「変数」に「値」を代入する、つまり「変数」という空の箱に、「値」という物を入れることができます。たとえば１０行目に「DATFILE=./data.xyz」とあるのは、「DATFILE」という変数（空き箱）に「./data.xyz」という値（物）を入れたことになります。ここではファイルの名前が定義されています。[20]

[20] この例では筆者らが用意した data.xyz を読み込んでいますが、CMAP が配布するオリジナルファイル cmap_mon_v0203_79.txt を読み込ませる場合、11 行目の#を消し、10 行目の先頭に#をつけます。

```
 1:#################################################################
 2:# Define Mapping Area
 3:#################################################################
 4:XMIN=0.00              #Horizontal minimum [degree]
 5:XMAX=360.00            #Horizontal maximum [degree]
 6:YMIN=-90.00            #Vertical minimum [degree]
 7:YMAX=90.00             #Vertical maximum [degree]
 8:XWID=21.0              #Width of image [cm]
 9:YWID=10.5              #Height of image [cm]
10:DXa=30.0               #a:Horizontal Annotation Interval [degree]
11:DXf=30.0               #f:Horizontal Frame Interval [degree]
12:DXg=10.0               #g:Horizontal Grid Interval [degree]
13:DYa=30.0               #a:Vertical Annotation Interval [degree]
14:DYf=30.0               #f:Vertical Frame Interval [degree]
15:DYg=10.0               #g:Vertical Grid Interval [degree]
16:D=2.5                  #grid size
17:#################################################################
18:# GMT Options
19:#################################################################
20:RFLAG=-R${XMIN}/${XMAX}/${YMIN}/${YMAX}
21:JFLAG=-JX${XWID}d/${YWID}d
22:BFLAG=-Ba${DXa}f${DXf}g${DXg}:Longitude:/a${DYa}f${DYf}g${DYg}:Latitude:neWS
```

変数 DYa の値で置き換え

第２ブロックに移りましょう。１行目から１６行目にかけては、引き続き変数に値が代入されています。ここで定義されているのは描画範囲です。それぞれの変数と値の意味は、図 3-2 を参考にしてください。

１７行目から２２行目で定義されているのは GMT で利用する-R, -J, -B オプションです。ここで「=」の右側に「${ }」という記号がありますね。これは変数に代入されている値で置き換えることを示します。例えば${XMIN}と書いてあったら、それを変数 XMIN に代入されている値で置き換えることを意味します。４行目で変数 XMIN には 0.00 という値が定義されていますから、-R${XMIN}/は-R0.00/に置き換えられるというわけです。

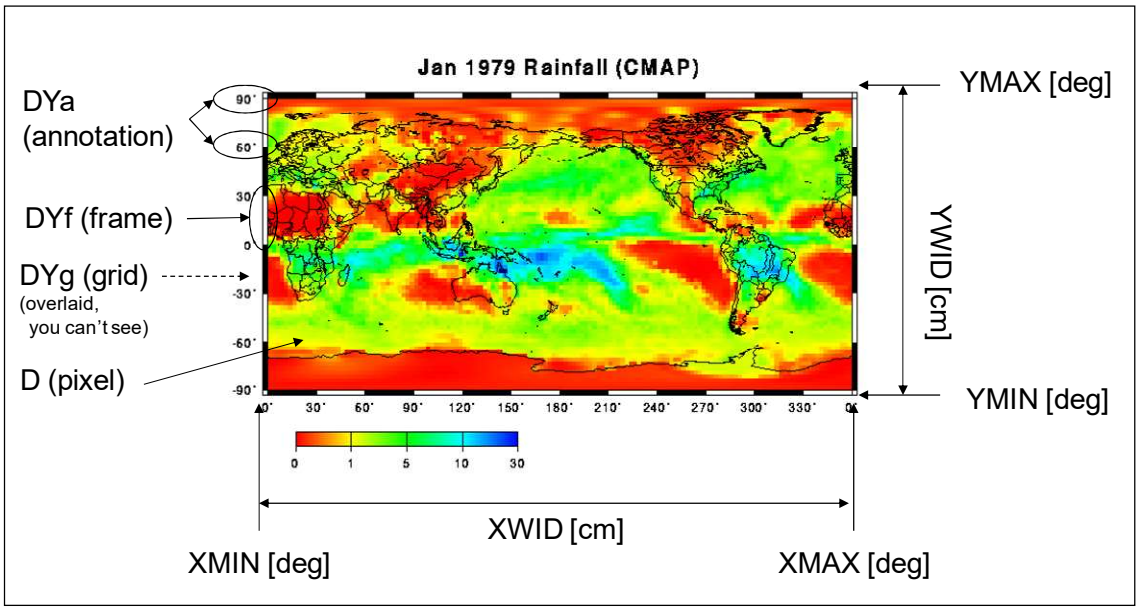


図 3-2 GMT の描画範囲

```
 1:#############################################################
 2:#---Jobs
 3:#############################################################
 4:#awk '($2==1){print $4, $3, $5}' $DATAFILE | \
 5:#xyz2grd     $RFLAG -G$GRDFILE -I${D}/${D} -F
 6:xyz2grd     $RFLAG -G$GRDFILE -I${D}/${D} -F $DATFILE
 7:#
 8:psbasemap   $RFLAG $JFLAG $BFLAG -X5.0 -Y5.0          -K  > ./$EPSFILE
 9:grdimage -O $RFLAG $JFLAG $GRDFILE -C$CPTFILE         -K >> ./$EPSFILE
10:pscoast  -O $RFLAG $JFLAG -Dc -W5 -N1 -I1             -K >> ./$EPSFILE
11:psscale  -O -C$CPTFILE -D5.0/-1.5/8/0.5h -L           -K >> ./$EPSFILE
12:pstext   -O $RFLAG $JFLAG -N                      << EOF >> ./$EPSFILE
13:180 110 24 0 0 1 6 Jan 1979 Rainfall (CMAP)
14:EOF
```


第３ブロックにはコマンドが書かれています。ここがこのシェルスクリプトの本体と言えるところです。[21] 6 行目が最初のコマンドです。$RFLAG とあるのは[22]、変数 RFLAG に代入された値で置き換えるということなので、この行は

「xyz2grd –R0.00/360.00/-90.00/90.00 –G./grd　–I2.5/2.5 –F」

と書いた場合と全く同じになります。ここで、xyz2grd がコマンドで、「-」（ハイフン）から始まる文字列がオプションです。第１章で「ls -l」というコマンドを勉強しましたが、ls というファイルを表示するコマンドに、-l というオプションを加えることで、詳細なリストを表示するように機能が拡張することができました。８行目の psbasemap 以降のコマンドではリダイレクト(>)がありますね。忘れてしまった使い方は、第２章を復習しましょう。

さて、第３ブロックではどのような作業が行われているのでしょう。概念図で表すと図 3-3 のようになります。

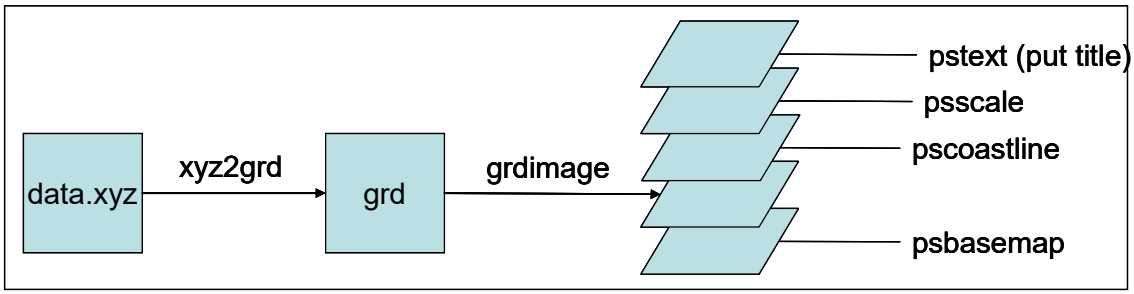


図 3-3　global.sh で行われる作業

まず、xyz2grd というのは「経度、緯度、データ」という書式で書かれた１地点ごとのテキストファイル（変数は DATFILE、値は data.xyz）を地図のような２次元情報を持つグリッドデータに変換する GMT コマンドです。この結果、中間ファイル（変数は GRDFILE、値は./grd）が作成されます。psbasemap、grdimage、pscoast、psscale、pstext が作図コマンドです。GMT は OHP シートやトレーシングペーパーを重ねるようにして作図していきます。psbasemap はその名の通り、図の下絵を描くコマンドです。作成した下絵

[21] 4,5 行目はコメント文になっていますが、前述した通り、オリジナルファイル cmap_mon_v0203_79.txt を読み込ませる場合は#を消して有効に、6 行目は先頭に#をつけて無効にします。
[22] ${RFLAG}と$RFLAG は同じです。「{}」は省略することができます。

は出力画像ファイル（変数は EPSFILE、値は image.eps）に書き込まれます。grdimage は xyz2grd で作った中間ファイルを描画するコマンドです。降水量に応じた色はここで設定され、出力画像ファイルに書き込まれます。pscoast は海岸線や国境線、河川を、psscale はカラーバーを、pstext は文字列（タイトルなど）を描くコマンドです。 x , y, size, angle, fontno, justify の順で引数を与えます。size はフォントの大きさ、angle は水平から反時計回りに計った角度、fontno はフォント番号（種類）、justify は文字列の配置をそれぞれ意味しています。これらの GMT コマンドが順々に出力画像ファイルに書き込まれていきます。

3.3 GMT コマンドのオプション

GMT コマンドでよく使うオプションを表 3-1 にまとめました。これらの詳しい使い方は、GMT のマニュアルを参照してください。すぐれた web サイトもいくつも公開されているので、google 等で検索してみてもよいでしょう。

ここで特に重要なのが「-O」と「-K」の２つのオプションです。GMT では OHP シートを重ねるようにして図を描きます。そのため、どの OHP シートが一番下で、どれが一番上かを示すことがとても重要です。GMT コマンドによる出力が、最初の OHP シートでない場合、オプション「-O」をつけなければなりません。「-O」は Overlay plot mode の略で、１枚目の OHP であることを示すヘッダー情報を省略することを示します。同様に、最後の１枚でない場合「-K」をつけなければなりません。「-K」は More PostScript code will be appended later の略で、最後の１枚であることを示すフッター情報を省略することを示します。

表 3-1 GMT コマンドのオプション

複数の GMT コマンドに共通するオプション	
-R	描画領域を指定する
-J	投影法を指定する
-B	軸書式を指定する
-O	最初の１枚目であることを示す header を省略する
-K	最後の１枚であることを示す trailer を省略する
psbasemap のオプション	
-X	x 方向に原点を移動する（単位：cm）
-Y	y 方向に原点を移動する（単位：cm）
pscoast のオプション	
-D	地図の解像度
-W	線の太さ
-N	国境線の描画
-I	河川の描画
psscale のオプション	

-D	カラーバーの位置と大きさを指定する
-L	カラーバーの間隔を等間隔にする
pstext のオプション	
-N	basemap の外側でも文字列を表示します。

3.4　CPT ファイル

図 3-1 では降水量が小さいところは赤、大きいところは青で示されています。それぞれの色と降水量との対応は左下のカラーバーに示されていますね。色が設定されているのは降水量が０から３０までの間で、赤、黄色、緑、シアン、青の５色が割り当てられていることが分かります。この色の設定をしているのが CPT (Color Palette File)ファイルです。図 3-1 で使われた CPT ファイルは「grad.cpt」です。

第１章で使った grad.cpt を開いてみましょう。図 3-4 のようになっているはずです。CPT ファイルは基本的に１行が８列の数値からなります。それぞれ、開始値、R、G、B、終了値、R、G、B の順です。例えば開始値０のとき赤（RGB で 255/0/0）を、終了値１のとき黄（RGB で 255/255/0）で表示し、その間を段階的に色を変えたい場合、

0　255　0　0　1　255　255　0

と書きます。このとき値の区切りには必ず tab を使ってください。space を使うとエラーになりますので注意して下さい。値が最小値（0）より小さい値、最大値（30）より大きい値、実数以外の値に色を割り当てるときはそれぞれ B (Background)、F (Foreground)、N（Not a number）の後に RGB を入力します。「grad.cpt」の場合、０より小さいときは濃い赤、３０より大きい時は濃い青の色が設定されています。

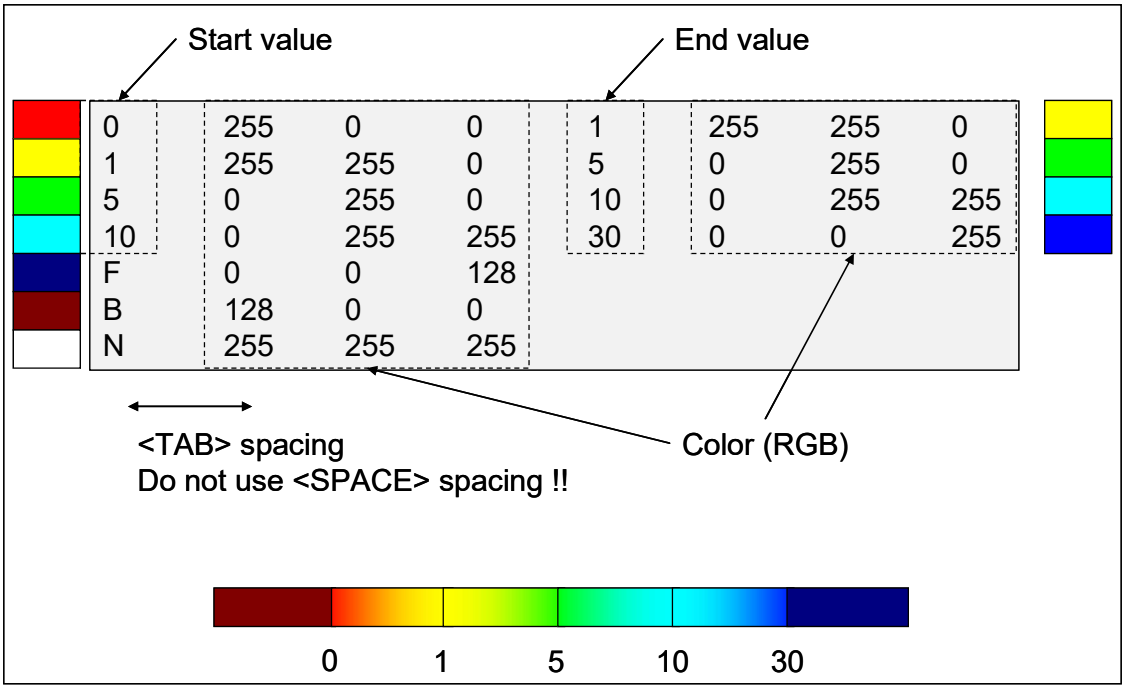


図 3-4　grad.cpt

宿題

global.sh のスクリプトを変更して、以下のような図を描いてみましょう。

1. 図のタイトルを変えましょう。（どのように変えても結構です。）
2. 図のカラーバーには単位が書いていませんね。カラーバーの適当なところに、単位[mm/day]を入れましょう。
3. 図を拡大してイランを拡大表示しましょう。水平方向は 60 度、鉛直方向は 30 度としてください。次に、イランは雨が少ないので、現在の CPT ファイル（grad.cpt）では、国内の雨の分布がはっきりしません。そこで、CPT ファイルを編集して、国内分布がはっきりするような図にしてください。最後に図のタイトルを変え、単位[mm/day]も入れましょう。

※宿題の解答例は、~/unix_semi/lesson3 にあります。

第 4 章

グリッドデータの処理(1)　FORTRAN プログラムを利用した演算

この章では CMAP データを用いて全球の降水量の解析をしてみましょう。CMAP のデータは全球、2.5°×2.5°解像度のグリッドデータです。また、1979 年から 2002 年まで、月単位のデータが配布されています。この章では、1987 年と 1988 年の年間降水量を比較した図を作図してみましょう。ちなみに、1987 年はエルニーニョだった年、1988 年はラニーニャだった年として知られています。

4.1　アスキとバイナリ

第２章で見た通り、CMAP データはテキストデータとして配布されています。配布されていたデータを全てダウンロードし、バイナリ（binary）という形式に変換したものが ~/unix_semi/dat_bin にあります。この章ではこのバイナリデータを使います。

計算機で情報を保存する際の方法は、基本的に２つしかありません。アスキかバイナリのどちらかです。アスキはテキストとも呼ばれます。アスキデータは Windows のメモ帳などで読むことができます。これと反して、バイナリは機械語なのでメモ帳では読めません。それぞれの特徴を表 4-1 にまとめました。

表 4-1 アスキとバイナリの比較

	アスキ	バイナリ
メモ帳で読めるか？	Yes	No
マシン依存性[23]	No	Yes
データサイズ	1 byte/文字	4byte/データ

アスキデータは一つの文字が 1byte で記録されますが[24]、バイナリデータでは、ひとつの実数を 4byte で記録することができます。例えば、実数「1.0」を保存するのに必要な記憶容量はアスキの場合 3 byte ですが、バイナリでは 4 byte になります。実数「1.000000」を保存するのに必要なバイト数は 8 byte ですが、バイナリではやはり 4 byte となります。

CMAP のデータは 2.5°×2.5°（経度×緯度）の解像度があるので、一月分のデータは 144×72 の実数で構成されることになります。このデータを図 4-1 のように横に 144 列、縦に 72 行並べて保存したとすると、ファイル容量は 144×72×4= 41472 バイトになります。実数であれば、値が 1.0 でも、1.0000 でも、9.87E+9 でも 4byte で表わされます。このため、~/unix_semi/dat_bin にあるどの年のどの月のデータも全て 41472 byte であることが分かるでしょう。ファイル名は CMAP203_19870100.cmap のように表されていますが、これは、（CMAP の ver2.03 の, 1987 年, 1 月の月単位）のデータであることを表していま

[23] たとえば、Sun Microsystems の Sparc が搭載されたマシンで作成したバイナリは、Intel の CPU が搭載されたマシンで作成されたバイナリと異なります。詳しく知りたい人は”big endian”あるいは”little endian”という語で web 検索してみてください。

[24] 英数の場合です。日本語のひらがなや漢字を扱うときは 2byte で記録されます。

す[25]。

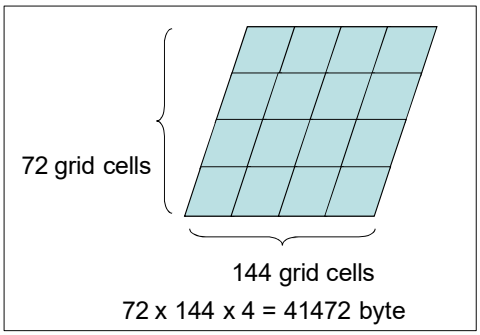


図 4-1　CMAP のバイナリデータ

4.2　月単位データの年単位データへの変換

さあ、本題に移りましょう。この章の目的は 1987 年と 1988 年の年降水量を比較することでした。今は月単位のデータしかありませんから、CMAP の 1987 年と 1988 年の月単位のデータを年単位のデータに変換し、その差をとることにしましょう。

まず年単位の降水量データを作成しましょう。やり方は図 4-2 に示すように１月から１２月までのデータを足し、最後に１２で割ればよさそうですね。

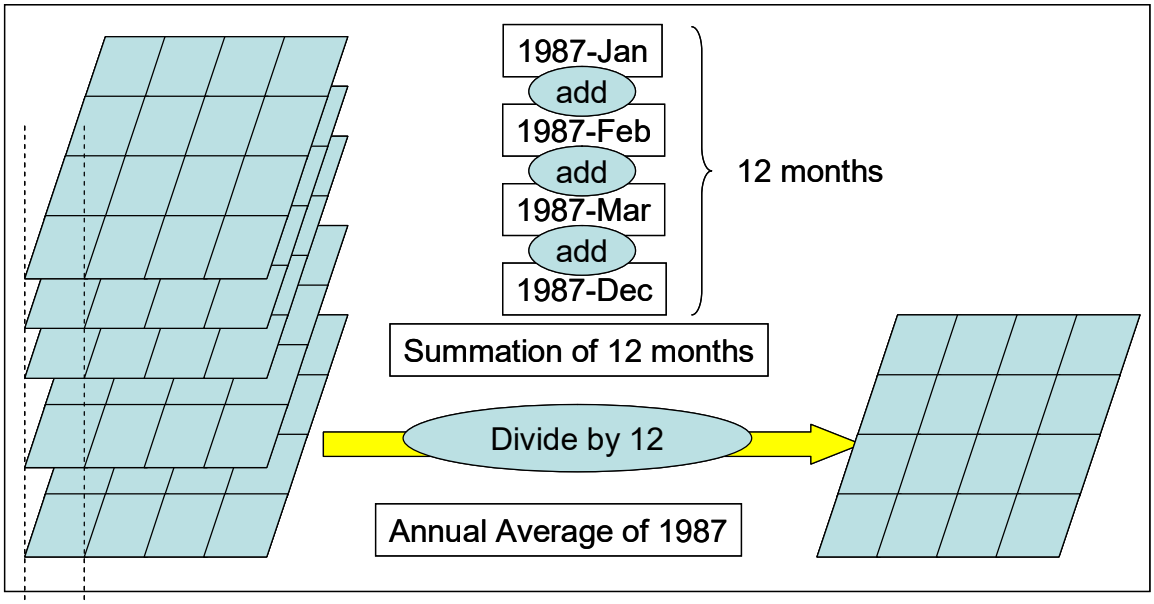


図 4-2　月単位から年単位へのデータ変換

このような作業を行うためにはバイナリファイル同士を足したり、割ったりする特別なコマンドが必要です。表 4-2 にあるようなコマンド[26]が使えるようになっています。これらのコマンドを使うときには、どのファイルを読んで、どのファイルに書き出すかなどを指定する必要があります。例えば temp.cmap という 144 列 72 行が実数 0 で満たされたバ

[25] 全球水資源モデル H08 の標準的な入出力データファイルもこの名前になっています。
[26] これらのコマンド群は H08 Analysis Tool と呼ばれ、全球水資源モデル H08 の標準的な入出力ファイルの処理に使われています。詳しくは H08 のマニュアルをご覧ください。

イナリファイルを新しく一つ作るときは、createcmap というコマンドを次のように使います。

```
% createcmap temp.cmap 0
```

このようにすると createcmap というコマンドに、temp.cmap と 0 という情報が渡されます。この temp.cmap や 0 のことを引数(argument)と言います。表 4-2 のコマンドに関しては、引数の順序を変えてはいけません。

表 4-2 バイナリファイルを操作するためのコマンド（その1）

プログラム	引数	機能
createcmap	OUT, num	num という実数で満たされたバイナリ OUT を作成する
addcmap	IN1,IN2,OUT	2つのバイナリ (IN1, N2)の和を求め、バイナリ OUT に書き出す
subcmap	IN1,IN2,OUT	2つのバイナリ (IN1, N2)の差を求め、バイナリ OUT に書き出す
mulcmap	IN, num, OUT	バイナリ(IN)に実数値(num)をかけ、バイナリ OUT に書き出す
divcmap	IN, num, OUT	バイナリ (IN)を実数値(num)で割り、バイナリ OUT に書き出す
procmap	IN1,IN2,OUT	2つのバイナリ (IN1, N2)の積を求め、バイナリ OUT に書き出す
ratcmap	IN1,IN2,OUT	2つのバイナリ (IN1, N2)の商を求め、バイナリ OUT に書き出す
asc2cmap	IN,OUT	テキスト(IN)をバイナリ(OUT)に変換する
cmap2asc	IN,OUT	バイナリ(IN)をテキスト(OUT)に変換する
xyz2cmap	IN,OUT	lon lat data 形式のテキスト(IN)をバイナリ(OUT)に変換する
cmap2xyz	IN,OUT	バイナリ(IN)から(lon lat data)のテキスト(OUT)に変換する
shiftcmap	IN,OUT	バイナリ(IN)を 180°東西方向にずらして、OUT に書き出す
upsidedowncmap	IN,OUT	バイナリ(IN)を南北逆転して、OUT に書き出す
cmap2eps	IN,CPT,OUT,[title1],[title2]	バイナリ(IN)を EPS 画像 (OUT)に変換する
eps2gif	IN,OUT [rot]	EPS 画像(IN)を他の画像フォーマット(OUT)に変換する。rot を付けると 90°回転する。
mon2yearcmap	DIR, PRJ, RUN, yearmin, yearmax, suf	月単位のデータを年単位に変換する
nofday	year mon	year 年 mon 月の日数を返す。

ここで、
IN: 入力するファイル、OUT: 出力されるファイル、CPT: cpt ファイル
num: 実数、year, mon: 年と月を表す整数
[title1], [title2]:タイトルの文字列。「[ ]」で囲まれた引数は省略が可能。
yearmin, yearmax: 処理開始年と処理終了年。西暦表示。4桁の整数
DIR: ファイルのあるディレクトリ（今いるディレクトリにファイルがあるときは./とする。）
PRJ: プロジェクト名を表す4文字の文字列（このテキストでは CMAP）
RUN: 実験名を表す4文字の文字列（このテキストでは 203_）
suf:拡張子(suffix)を表す文字列（このテキストでは.cmap）

それでは、表 4-2 にあるコマンドを使って作業を進めましょう。まず、ホームディレクトリに BINARY というディレクトリを作り、そこに 1987 年から 1988 年のバイナリデータをコピーしましょう。

```
% mkdir ~/BINARY
% cd ~/BINARY
% cp ~/unix_semi/dat_bin/*1987* .
% cp ~/unix_semi/dat_bin/*1988* .
```

ただ、実際は~/unix_semi/dat_bin/に既に 1987 年と 1988 年の年平均ファイルが用意されています。それも一緒にコピーしてしまったので、消しておきましょう。

```
% rm CMAP203_19870000.cmap
% rm CMAP203_19880000.cmap
```

それでは図 4-2 に示されたように、作業を進めましょう。まず 1987 年の年平均値ファイル「CMAP203_19870000.cmap」を新しく作ります。

```
% createcmap CMAP203_19870000.cmap 0
```

この「CMAP203_19870000.cmap」は 144 列×72 行の実数からなるバイナリファイルで今は、すべてに０の値が入っています。このファイルに、1987 年の１月のデータを足しましょう。

```
% addcmap CMAP203_19870000.cmap CMAP203_19870100.cmap CMAP203_19870000.cmap
```

次にこのファイルに２月のデータも足します。

```
% addcmap CMAP203_19870000.cmap CMAP203_19870200.cmap CMAP203_19870000.cmap
```

これで、１月と２月の和が CMAP203_19870000.cmap に入ったことになりますね。これを１２月まで続けていきます。

```
% addcmap CMAP203_19870000.cmap CMAP203_19871200.cmap CMAP203_19870000.cmap
```

最後に１２で割りましょう。

```
% divcmap CMAP203_19870000.cmap 12 CMAP203_19870000.cmap
```

これで、1987 年の平均値を得ることができました。同様に 1988 年の平均値を作ってみましょう。

さて、12 カ月分のファイルを足して、最後に 12 で割ることで、年平均を求めてきましたが、このやり方は実は正確ではありません。なぜなら各月の日数が異なるからです。正確に計算するには、

```
% mulcmap CMAP203_19870100.cmap 31 temp.cmap
```

のように降水量に各月の日数をかけたのち、

```
% addcmap CMAP203_19870000.cmap temp.cmap CMAP203_19870000.cmap
```

として足し合わせ、最後に

```
% divcmap CMAP203_19870000.cmap 365 CMAP203_19870000.cmap
```

と 1 年の日数で割ると正しい計算になります。しかし毎月の日数や閏年を考慮するのは大変です。そこで月単位のデータを年単位のデータに正確に変換する mon2yearcmap というコマンドがあります。以下のようにすると、1987 年から 1988 年までの年平均値を正確に計算することができます。

```
% mon2yearcmap ./ CMAP  203_ 1987  1988  .cmap
```

4.3　データの差と図化

それでは、1987 年と 1988 年の差をとってみましょう。図 4-3 に示されたように 1987 年のデータから 1988 年のデータを引いて、結果を dif.cmap というファイルに書き出してみましょう。

```
% subcmap CMAP0203_19870000.cmap CMAP0203_19880000.cmap dif.cmap
```

この結果を地図にしましょう。ここで差を表示するための「dif.cpt」という CPT ファイルを自分で作ってみてください。cmap2eps コマンドの機能を使って Difference という主タイトルと 1987-1988 という副タイトルをつけましょう。

```
% cmap2eps dif.cmap dif.cpt dif.eps Difference 1987-1988
```

図 4-3 にあるような差が見られましたか？

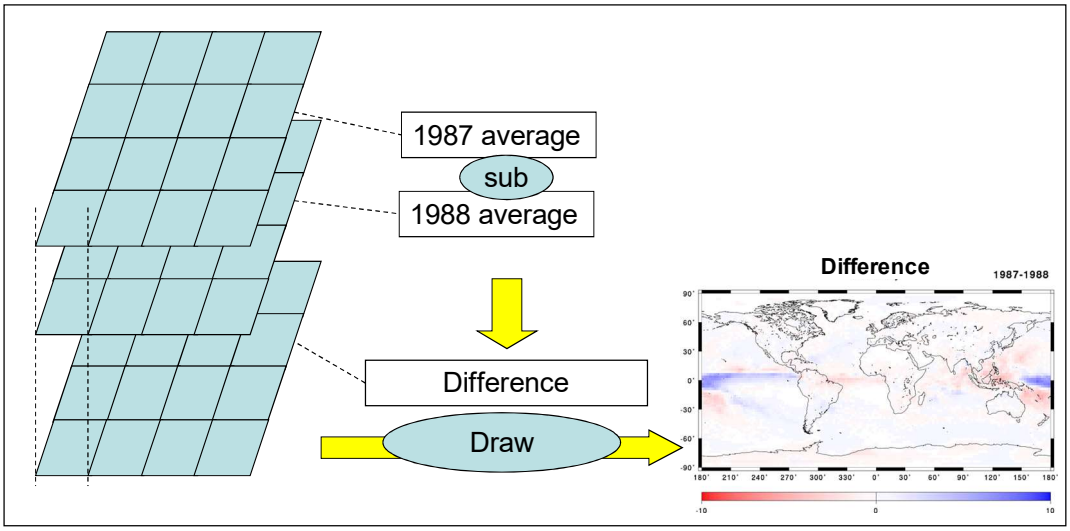


図 4-3　1987 年と 1988 年の差

宿題

1. 1987 年と 1988 年の JJA (June- July- August) と DJF (December- January- February)で降水量データを作成し、違いを表す図を作りましょう（自分のオリジナルの CPT ファイルを作りましょう）。ここで 1987 の DJF とは 1986 年 12 月から 1987 年 2 月まで、1988 年の DJF とは 1987 年 12 月から 1988 年 2 月までを示すものとします。
2. 例題のやり方だともともとの雨が多いところ（熱帯の海洋など）で、差が目立つような図になってしまいます。たとえば、もともと雨の少ないサハラ砂漠では差がないように見え、雨の多い熱帯で差が大きく見えますが、前者は比較的小さい値から小さい値を引いていて、後者は比較的大きい値から大きい値を引いているためにおこります。そこで、1987 年と 1988 年の降水量の差を 1987 年の降水量に対する割合として比較する図を描きましょう。つまり、1987 年の年平均値が 5mm/day で、1988 年が 6mm/day なら、(5-6)/5=-0.2 となるようにしましょう。オリジナルの CPT ファイルをつくり、図示してみましょう。

※宿題の解答例は~/unix_semi/lesson4 にあります。

※ 出力結果がどうしても異常になる場合は、Appendix D を参照し、 Endian を変更してみてください。

第 5 章

FORTRAN(1) FORTRAN の入出力

第4章では、表 4-2 のコマンドを利用してバイナリファイルの計算を行いました。これらのコマンドは FORTRAN というプログラム言語を利用して筆者が自作したものです。この章では FORTRAN を使ってどうやってファイルを読み、書き出すのかについて説明します。

FORTRAN は 1950 年代に開発され、その後時代に合わせて書式や機能が変化してきました。FORTRAN は時代遅れのプログラミング言語と思われがちですが、科学の分野ではまだ広く用いられています。とくに、水文気象学の分野で重要なプログラムである、気候モデル、陸面過程モデル、河川モデルなどは世界的に見ても、FORTRAN で書かれることが多いようです。

なお、このテキストでは FORTRAN を使いますが、自分でプログラミングをするときには、C、C++、C#、Java、Ruby など、好きなプログラミング言語を使ってかまいません。また、この章では FORTRAN のごく限られた、また少々特殊な使い方しか紹介できません。きちんと勉強したい人は、FORTRAN の教科書を読んでください。

5.1 FORTRANソースコードの基本

FORTRAN のソースコードを見てみましょう。これは端末に「Hello, World」などの文字を出力するプログラムです。

```
1:     program hello
2:c
3:c this is comment.
4:c
5:     write(6,*) 'Hello, World'
6:     write(6,*) 'A very very long sentence such as this
7:    $     must be divided in two lines'
8:c
9:     end
```

１行目では「hello」という名前のプログラムが始まることが宣言されています。「program」というのが開始宣言の命令で、FORTRAN のソースコードの１行目に必ず書かなければなりません。FORTRAN のソースコードで本文を書けるのは７桁目から７２桁目までです。この例でも１行目は行頭からスペースを６つ空けて７桁目から「program hello」と書き出されています。２文字目から５文字目には「行番号」という特別な意味を持つ整数を入れることができるのですが、現在はほとんど使われません。使い方を知りたい人は FORTRAN の教科書を参照してください。

２行目から４行目はコメント文です。コメントを書きたい場合には、１桁目に c または*を入力します。ということになります。

５行目には「Hello, World」と端末に書き出す命令が書かれています。「write」というのが書き出しの命令です。括弧内の最初にあるのは「装置番号」を、次にあるのは「書式」を表します。装置番号とは書き出し先を示す正の整数です。装置番号はユーザが定義するのですが、5 と 6 だけはそれぞれ標準入力（キーボード）、標準出力（端末）と決ま

っています。書式が「*」というのは自動書式という意味です。その後「”」で囲まれた文字列が出力される文字列となります。

６行目から７行目には「A very very long sentence such as this must be divided in two lines」と端末に書き出す命令が書かれています。この場合、72 桁目までに命令を書ききれないので２行に分けます。２行にわたって命令を書きたい場合は、続きの行の６桁目に０以外の文字を入力します。この例では「$」を用いています。

5.2 FORTRANソースコードのコンパイル

それでは練習のために、まずこの例と全く同じテキストファイルを emacs で作成し、example1.f という名前で保存して下さい。さて、このファイルはこのままでは実行できません。FORTRAN コンパイラという特別なコマンドを利用して、実行可能な機械語に変換する必要があるのです。この作業を「FORTRAN ソースコードをコンパイルする」といいます。Appendix C に従ってあなたの UNIX コンピュータが設定されている場合、gfortran という FORTRAN コンパイラが使えるようになっているはずです。FORTRAN ソースコード example1.f をコンパイルするには次のようにします。

```
% gfortran example1.f
```

すると、a.out という実行可能ファイルが作成されます。これを

```
% a.out
```

のように実行すると、

```
Hello, World
A very very long sentence such as this must be divided in two lines
```

のように端末に表示されるはずです。ソースコードをコンパイルしてできるファイルの名前は全て a.out となります。ただ、これではどのソースコードをコンパイルしたか後で分からなくなります。そこで「-o」というオプションをつけ、その後にファイル名を入れると、実行ファイル名を変えることができます。例えば、

```
% gfortran example1.f –o example1
```

のように、とすると、example1 という実行ファイルができます。これは gfortran 以外の多くの FORTRAN コンパイラで有効なようです。

5.3 FORTRANのファイルの入出力

第４章の表 4-2 で紹介したコマンドの一つに cmap2asc があります。このプログラムは、バイナリデータをアスキデータに変換するコマンドでした。例えば、第４章で使った 1987 年 1 月のバイナリファイルを temp.txt という名前のテキストファイルに変換したい場合は、

```
% cd ~/BINARY
% cmap2asc CMAP203_19870100.cmap temp.txt
```

とするのでしたね。

この cmap2asc のソースコードが~/unix_semi/src/cmap2asc.f にあります。

この節では、このプログラムのソースコードを４つのブロックに区切って、解説していきます。

```
 1:     program cmap2asc
 2:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 3:cto   Convert binary (144x72) to ascii
 4:con   30th/May/2002
 5:cby   nhanasaki
 6:cat   IIS, UT
 7:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 8:      implicit none
 9:c
10:      integer          nx
11:      integer          ny
12:      parameter       (nx=144)
13:      parameter       (ny=72)
14:c
15:      integer          i,j
16:      real            rdat(nx,ny)
17:      character*128    cifname
18:      character*128    cofname
19:c
20:      integer          iargc
```

第１ブロックでは、まず１行目でプログラムの開始を宣言しています。「program ○○○」と書くことで、このコードは○○○というプログラムだということが宣言できます。

２行目から７行目にかけてはコメント文で、作成者や日付などが記録されています。

８行目には「implicit none」と書いてありますが、これは「暗黙の変数型」[27]を使わないことを意味するもので、忘れずに書く習慣を付けてください。

１０行目から２０行目にかけて記述されているのが変数の型の宣言です。第３章では Bourne シェルスクリプトの変数と値の代入について勉強しました。FORTRAN でも同じように変数に値を代入することができますが、より細かく設定を行うようになっています。ここでは、変数（第３章で空き箱だといいました）を宣言しています。integer は整数型、real は実数型、character は文字型の宣言に用います。変数の宣言は図 5-1 のように「数字や文字の入る箱の種類や大きさを決めること」と考えるとわかりやすいでしょう。たとえば「integer nx」というのは、整数(integer)をひとつだけ入れることができる nx という箱を用意することです。「character*128 cifname」は 128 文字までの文字列(character*128)を入れることのできる cifname という箱を用意すること、といった具合です。rdat(nx,ny)というのは nx 列 ny 行に区切られた実数を格納できる箱だと考えてください。こうした２つ以上の値を代入できる変数を配列（array）といいます。

１２行目と１３行目にある parameter は変数に値を定数として代入します。この値は定

[27] 「暗黙の変数型」とは、変数の型の宣言をしなくても i～n から始まる変数は整数型、それ以外を実数型とみなす、などという FORTRAN の仕様。一見変数宣言を省けるので便利に見えるが、思わぬ誤動作やバグの原因となりやすいので使わないほうがよい。

数となり、プログラム実行中は書き換えることはできません。ここで nx と ny は配列 rdat の宣言で用いられています。この場合、nx と ny は必ず定数として値が代入されていなければなりません。

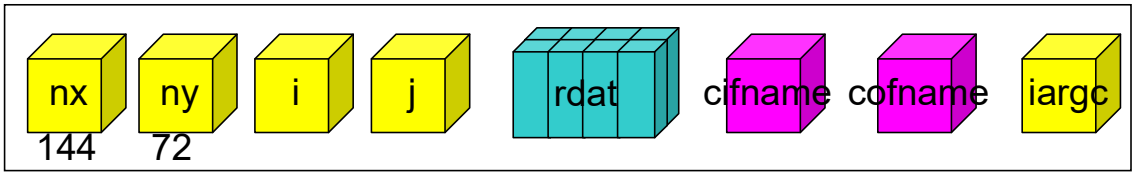


図 5-1　FORTRAN の変数の型

```
 1:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 2:c Get file name
 3:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 4:      if (iargc().ne.2) then
 5:        write(6,*) 'Usage: cmap2asc Infile Outfile'
 6:        stop
 7:      end if
 8:c
 9:      call getarg(1,cifname)
10:      call getarg(2,cofname)
```

第２ブロックでは、ファイル名を読み込んでいます。

まず４行目ですが、if の直後にある括弧内の条件文を判定し、真である場合は次の行にある命令を実行します。条件文の書き方は表 5-1 の通りです。ここで「iargc()」はコマンド（この例の場合 cmap2asc）に与えられた引数の数を返す関数です。よって、ここでの条件文は「引数の数が 2 でなければ真」ということになります。真の場合、次の行の命令が実行されます。偽の場合何も実行されず、end if 以降の命令を実行します。

５行目の write 文は既に勉強しました。括弧の中で装置番号と書式を指定するのでしたね。装置番号の「6」は端末を表します。書式の「*」は自動設定を表します。どういう書式で書き出すかは細かく指定できるので、知りたい人は FORTRAN の教科書を参照してください。書き出したい文字列は「 ‘’ 」の間に書きます。６行目の「stop」はプログラムの終了を意味します。つまり、４行目から７行目を普通の言葉で書くと、「もし引数の数が 2 でなかったら、画面に'Usage: cmap2asc Infile Outfile'とデフォルトの書式で端末に出力してコマンドを終了します。それ以外の場合（つまり引数の数が 2 なら）、作業を継続します。」となります。

９行目、１０行目はそれぞれ１番目と２番目の引数を読み込んでいます。１番目の引数を読み、変数 cifname に代入するのが call getarg(1,cifname)という命令です。表 4-2 を見ると cmap2asc には入力ファイル名と出力ファイル名の２つの引数が必要ですね。９行目と１０行目を普通の日本語にすると、「１番目の引数を変数 cifname に代入します。２番目の引数を変数 cofname に代入します。」となります。

表 5-1　条件文の種類

a.gt.b	a is greater than b （a が b より大きいとき真）
a.ge.b	a is greater equal than b （a が b 以上のとき真）

a.lt.b	a is less than b （a が b より小さいとき真）
a.le.b	a is less equal than b （a が b 以下のとき真）
a.eq.b	a is equal to b （a が b に等しいとき真）
a.ne.b	a is not equal to b （a と b が等しくないとき真）

```
1:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
2:c Read Data
3:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
4:      open(15, file=cifname, access='direct', recl=nx*4)
5:      do j=1, ny
6:        read(15,rec=j) (rdat(i,j),i=1,nx)
7:      end do
8:      close(15)
```

第３ブロックでは、バイナリファイルから値を読み込みます。まず４行目でファイルを開いています。open 文は「open(装置番号, file=ファイル名, access='アクセス方法', recl=レコード長)」という形で使います。

装置番号は既に説明した通り、操作する対象を示す正の整数です。ここでは開こうとしているファイルに対して割り当てる整数です。アクセス方法は'direct'と'sequential'しかありません。前者はバイナリファイルを、後者はアスキファイルを読み取ります。レコード長は Access='direct'でバイナリファイルを読み込むときだけに設定が必要な正の整数です。レコード長とは１回 read（write）するときに何 byte のデータを読み取る（書き出す）かを指定します。CMAP は図 5-2 のように nx 列 ny 行からなる２次元のデータで、j 行目には nx 列の実数が収納されています。１つのデータは 4byte なので、レコード長は nx*4 byte となります。

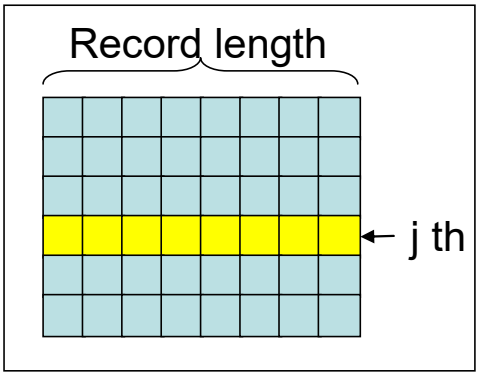


図 5-2　CMAP のバイナリデータとレコード長

５行目は「do ループ」と呼ばれる繰り返し処理を示します。ここでは j を 1 から ny まで１ずつ増加させながら、「do」と「end do」の間にある命令を ny 回繰り返すことを示します。

６行目はファイルから変数を読み込んでいます。read 文は「read(装置番号,読み取るレコード) 読み込んだ値を代入する変数」という形をしています。 (rdat(i, j), i=1, nx)は i を 1 から nx まで１ずつ増加させて nx 回繰り返して rdat に読み込むという意味です。

８行目はファイルを閉じています。close 文は「close(装置番号)」のように使います。このように操作が終わったら開いたファイルを必ず閉じましょう。

第３ブロックの内容を普通の言葉で書くと、以下のようになります。

［１～３行目］（コメント文）

［４行目］ファイル cifname を開き、装置番号として 15 を割り当てます。このファイルはバイナリで書いてあり（直接アクセス）、レコードの長さは nx*4(=144*4=576)バイトです。

［５行目］j を 1 から ny まで変化させます。（つまり end do 間での処理を ny 回繰り返します）

［６行目］装置番号 15 を割り当てたファイルの j 番目のレコードを読み取ります。データを配列 rdat に入力します。

［７行目］end do まで来たら、do に戻って繰り返します。

［８行目］装置番号 15 を割り当てたファイルを閉じます。

```
 1:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 2:c Write Data
 3:ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 4:      open(16, file=cofname, access='sequential')
 5:      do j=1, ny
 6:        write(16,*) (rdat(i,j), i=1,nx)
 7:      end do
 8:      close(16)
 9:c
10:      end
```

第４ブロックは、変数 rdat に代入された値をアスキで書き出す操作をしています。４行目の open 文でファイル cofname を開きます。装置番号には 16 を割り当て、アクセス方法は sequential を設定します。アクセス方法が direct ではないのでレコード長は指定する必要がありません。

第４ブロックを普通の言葉で書くと、次のようになります。

［１～３行目］（コメント文）

［４行目］装置番号として 16 を割り当て、ファイル cofname を開きます。このファイルはアスキで書かれています。

［５行目］j を 1 から ny まで変化させます。

［６行目］装置番号 16 に配列 rdat を書き出します。i は 1 から nx まで変化させます。書式はデフォルトです。

［７行目］end do まで来たら、do に戻って繰り返します。

［８行目］ファイル 16 を閉じます。

［１０行目］プログラムを終了します。

宿題

1. 表 4-2 にある addcmap の FORTRAN ソースコードを読んで、普通の言葉に直してみましょう。ソースコードは~/unix_semi/src/addcmap.f にあります。
2. cmap2asc.f と反対の機能を持つプログラム、asc2cmap.f を作りましょう。（編集が終わったら、コンパイルして実行してみましょう）

※宿題の解答例は~/unix_semi/lesson5 にあります。

第 6 章

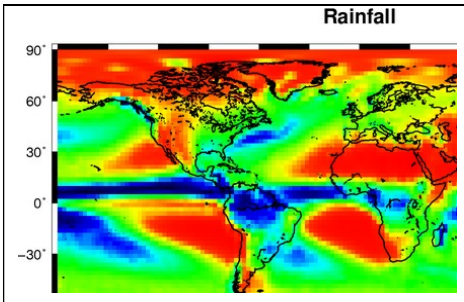
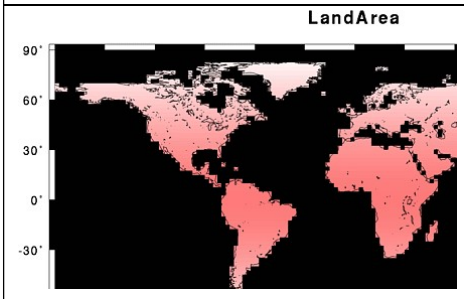
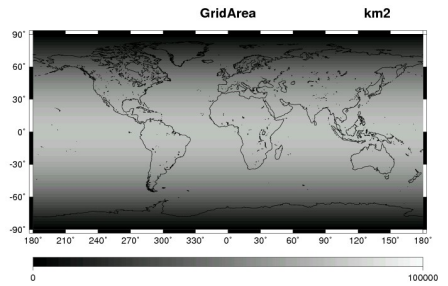
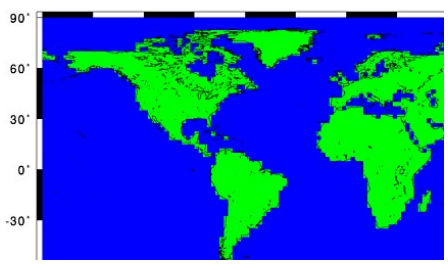
グリッドデータの処理(2)　FORTRAN プログラムを使ったマスク処理

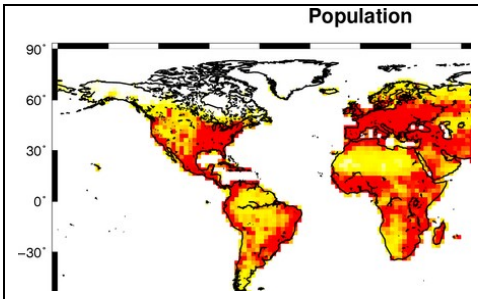
東京の年降水量はどのくらいでしょうか。また、世界の年平均降水量はどのくらいでしょうか。CMAP データを使って調べてみましょう。世界のどのくらいの地域で、年降水量が東京を上回るのでしょうか。また、どのくらいの人が 500mm/year より降水量の少ない場所に住んでいるのでしょうか。

6.1　この章で利用するデータ

この章では表 6-1 のデータを用います。データは~/unix_semi/lesson6 にあります。

表 6-1　第6章で利用するデータ（バイナリファイル）

	**年平均降水量（1979–2001）** ファイル名：prc.cmap 単位：mm/year 解像度：全球／2.5°×2.5° 備考：CMAP データを使って作成したもの
	**グリッド面積（陸のみ）** ファイル名：lndara.cmap 単位：km^2 解像度：全球／2.5°×2.5°
	**グリッド面積（海陸）** ファイル名：grdara.cmap 単位：km^2 解像度：全球／2.5°×2.5°
	**海陸マップ** ファイル名：lndmsk.cmap 単位：なし 解像度：全球／2.5°×2.5° 備考：陸が 1、海が 0



人口
ファイル名：pop.cmap
単位：人
解像度：全球／2.5°×2.5°
出典：CIESIN GPW2[28]

6.2　CMAP のバイナリファイルを操作するためのコマンド

第 4 章では CMAP のバイナリファイルを操作するためのコマンドについて勉強しましたね。ここではさらに多くのコマンドを説明します（表 6-2）。

表 6-2　利用可能なコマンド（その 2）

プログラム	引数	機能
avecmap	IN, miss	バイナリ IN の miss 以外のデータの平均値を計算する
maxcmap	IN, miss	バイナリ IN の miss 以外のデータの最大値をみつける
mincmap	IN, miss	バイナリ IN の miss 以外のデータの最小値をみつける
sumcmap	IN, miss	バイナリ IN の miss 以外のデータの総和を計算する
pointcmap	IN, lon, lat	経度が lon、緯度が lat の場所のデータを表示する
findcmap	IN, sign, num, OUT	バイナリ IN のデータで num と sign するものを表示し、バイナリ OUT に書き出す。それ以外は 0 が入る。
rplccmap	IN, sign, num1, num2, OUT	バイナリ IN のデータで num1 と sign するものを num2 に置き換えてバイナリ OUT に書き出す。それ以外は IN と同じデータが入る。
maskcmap	IN, MASK, sign, num, OUT	バイナリ MASK のデータが num と sign するとき、バイナリ IN のデータをバイナリ OUT に書き出す。それ以外は 0 が入る。
maskrplccmap	IN, MASK, sign, num1, num2, OUT	バイナリ MASK のデータが num1 と sign するとき、バイナリ IN のデータを num2 に置き換えて OUT に書き出す。それ以外は IN と同じデータが入る。
IN, MASK, OUT はそれぞれバイナリファイル sign は条件を示し、eq, ne, gt, ge, lt, le から一つを選択できる。それぞれ「と一致する」、「と一致しない」、「より大きい」、「以上の」、「より小さい」、「以下の」となる。 num, num1, num2 は実数、lon, lat は経度と緯度を表す実数 miss は実数の欠損値[29]		

[28] http://sedac.ciesin.columbia.edu/gpw/

[29] 欠損値(missing value)とは実数であるが計算処理の対象から外す特殊な数を示す。例えば、ゼロか正の数しか取りえないデータ（例えば人口や降雨など）に-999 などの値を与えて、データが欠損していることを明示するために使う。気象分野では 1.0E20 がよく使われる。ただし、CMAP データの場合、全期間・前グリッドにわたって欠損のないデータであり、欠損値は割り当てられていない。

表 6-2 にある findcmap, rplccmap, maskcmap, maskrplccmap は少しわかりにくいので、例を挙げて説明します。まず、findcmap は条件に合ったデータを抽出するコマンドです。図 6-1(a) にある例を考えましょう。左図のように 1 から 16 までのデータが入っているバイナリファイル IN から、８より大きい（greater than）数字を抽出して結果をバイナリファイル OUT に書き出すには以下のようにします。

```
% findcmap IN gt 8 OUT
```

次に rplccmap は条件に合ったデータを別の実数に置き換えるコマンドです。ここで、条件に合わないデータは元の実数がそのまま出力されます。図 6-1(b) にある例のように、１から 16 までのデータが入っているバイナリファイル IN から、８より大きい（gt）数字を 1 で置き換えて結果をバイナリファイル OUT に書き出すには以下のようにします。

```
% rplccmap IN gt 8 1 OUT
```

maskcmap は別に用意したバイナリファイル MASK（図 6-1(c) に示されるものでマスクファイルと呼ばれる）が条件を満たすか判定し、条件を満たす場合はバイナリファイル IN のデータを抽出し、バイナリファイル OUT に書き出すコマンドです。図 6-1(d) にある例のように、1 から 16 までのデータが入っているバイナリファイル IN から、マスクファイル MASK が 1 と一致する(eq)ときだけデータを抽出してバイナリファイル OUT に書き出すには以下のようにします。

```
% maskcmap IN MASK eq 1 OUT
```

最後に maskrplccmap は別に用意したバイナリファイル MASK が条件を満たすか判定し、条件を満たす場合はバイナリファイル IN のデータを別の実数に置き換えて、バイナリファイル OUT に書き出すコマンドです。図 6-1(e) にある例のように、1 から 16 までのデータが入っているバイナリファイル IN から、マスクファイル MASK が 1 と一致する(eq)ときだけデータを 1 に変換して、結果をバイナリファイル OUT に書き出すには以下のようにします。

```
% maskrplccmap IN MASK eq 1 1 OUT
```

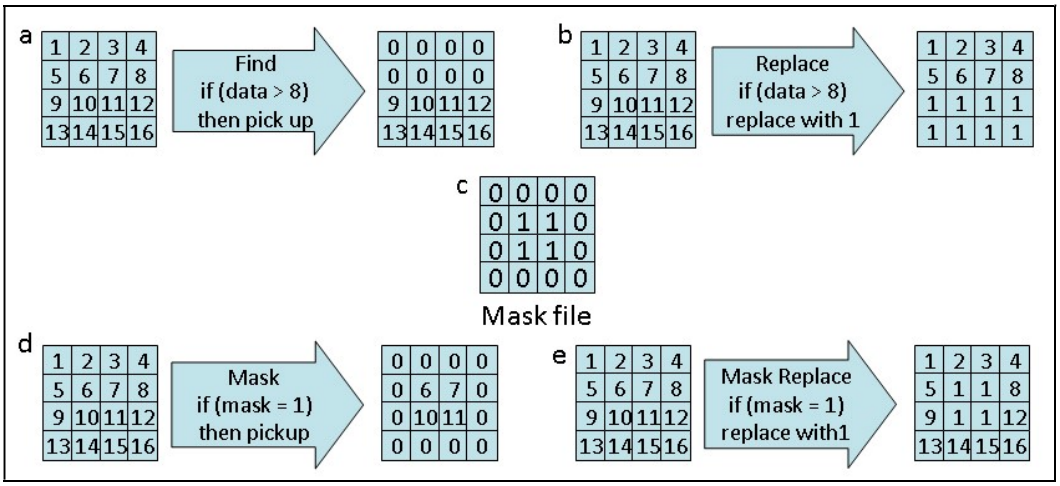


図 6-1 findcmap, rplccmap, maskcmap, maskrplccmap の概念

【例 1】

東京（東経 138.75°　北緯 36.25°[30]）での年降水量を求めましょう。

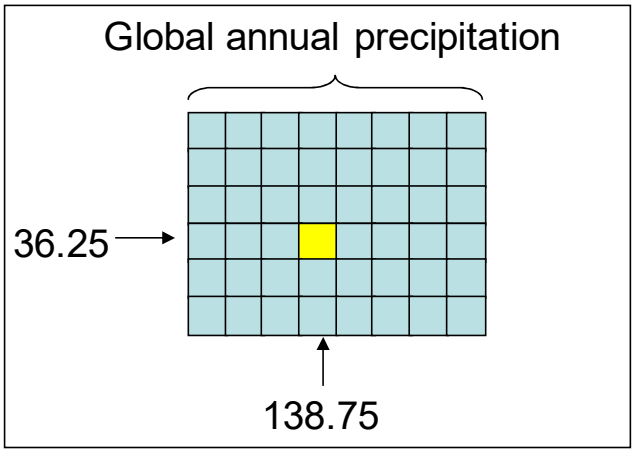


図 6-2　pointcmap の概念

まず表 6-1 にあるどのデータを使えばよいでしょう？また表 6-2 にあるコマンドのどれを使えばよいでしょう？年平均降水量データ(prc.cmap)を pointcmap で処理するのがよさそうですね。以下のようにコマンドを打ち込みましょう。

```
% pointcmap prc.cmap 138.75 36.25
```

このようにすると、図 6-2 に示すように、バイナリファイル prc.cmap を読み取り、東京付近のグリッド(北緯 36.25,東経 138.75)のデータを取り出すことができます。興味のある人は、~/unix_semi/src/pointcmap.f を見て、どのようなソースコードになっているのか調べてみてください。

【例 2】

世界の年平均降水量を求めましょう。

次に、全球の年平均降水量を求めましょう。表 6-1 にあるどのデータと表 6-2 にあるどのコマンドを使えばよいでしょう？ 年平均降水量データ(prc.cmap)を avecmap で処理するのがよさそうですね。以下のようにコマンドを打ち込みましょう。

```
% avecmap prc.cmap -999
```

このようにすると、バイナリファイル prc.cmap の全てのデータを足しあわせ、データの数で割って平均値を計算することができます。このとき欠損値（上の例では-999）はデータの足し合わせから除外されます。実はこの計算は正確ではありません。このことについては、この章の宿題で考えてみましょう。

[30] この緯度と経度が表す場所は、実際には群馬県高崎市です。CMAP はひとつのグリッドが大きいので（赤道付近で約 280km x 280km）、ここが東京から一番近いグリッドなのです。

【例３】

世界のどのくらいの地域で、年降水量が東京を上回るのでしょうか。面積を求めましょう。

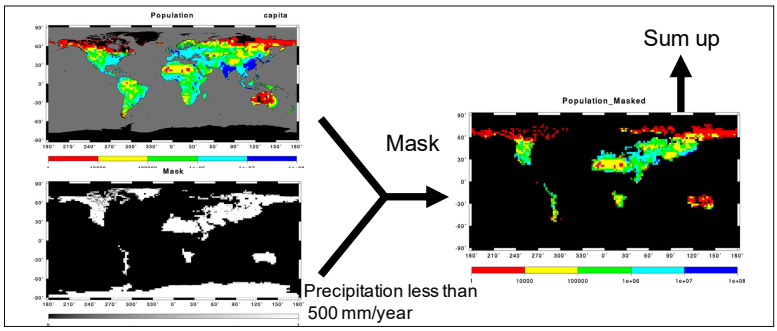


図 6-3　例4の計算方法

東京の年降水量は例 1 で求めたように 1530mm/year でしたね。そこでまず陸域の面積データ(lndara.cmap)を、年降水量データ(prc.cmap)が 1530mm/year 以上の場合だけ抜き出しましょう。

```
% maskcmap lndara.cmap prc.cmap ge 1530 temp.cmap
```

次に、抽出した面積データの合計を求めましょう。

```
% sumcmap temp.cmap 0
```

【例４】

年降水量が 500mm/year より少ない地域に住んでいる人口を求めましょう。

例３と同じように、人口データ（pop.cmap)を、年降水量データ(prc.cmap)が 500mm/year 未満の場合だけ抜き出しましょう（図 6-3）。

```
% maskcmap pop.cmap prc.cmap lt 500 temp.cmap
```

抜き出したグリッドの人口の合計を求めましょう。

```
% sumcmap temp.cmap 0
```

宿題

1. 世界の年平均降水量を正しく計算しましょう。例 1 では全てのデータの単純な平均を取る avecmap コマンドを利用して年平均降水量を求めました。しかし、地球は球体ですので、高緯度ほど各グリッドの面積が小さくなっています（表 6-1 の面積データを見ましょう)。そこで、面積で重みづけした平均値を求めましょう。まずどのようにデータを処理すればよいのか考えましょう。つぎに、表 6-2 にあるコマンドを組み合わせて処理を実行しましょう。ヒントは表 6-1 にある地表面面積データ(grdara.cmap)と表 4-2 にあったバイナリファイル同士の掛け算を行う procmap をうまく使うことです。なお、途中である数値をある数値で割り算しないといけないのです

が、ここは電卓などを使って手計算を行ってください。

2. 陸上の年平均降水量を次の表のように分類した場合、それぞれの地域はどれくらいの面積になるのでしょうか。また、それぞれの地域にどれくらいの人が住んでいるのでしょうか。次の表を埋めてみましょう。

Precipitation (mm/year)	Population (capita)	Area (km2)
0-50		
50-100		
100-500		
500-1000		
1000-2000		
2000-		
Total		

※宿題の解答例は~/unix_semi/lesson6 にあります。

第 7 章

FORTRAN (2) FORTRAN の時系列データの入出力

第7章では、時系列のバイナリファイルを効果的に処理するための FORTRAN プログラム技術について説明します。なお第5章でも述べましたが、基本的な FORTRAN の文法については自習してください。また、この章で紹介するソースコードは~/unix_semi/src/にあるので、必要に応じて参照してください。

7.1 FORTRAN の時系列ファイルの入出力

第４章では月単位の CMAP バイナリファイルを年単位に変換することを学びました。この処理を行うため、mon2yearcmap というコマンドを利用しました。mon2yearcmap コマンドは引数として与えられた開始年から終了年までの各年の１月から１２月のデータを読み、各年の平均値データを出力します。このとき、月の日数の違いを考慮するため、月毎のデータを足し合わせるとき、各月のデータに各月の日数をかけておき、最後に１年の日数（365 日、うるう年なら 366 日）で割る計算を行います。このソースコードが~/unix_semi/src/mon2yearcmap.f にあるので開いてみましょう。この章ではこのソースコードを４つのブロックに分けて解説していきます。

```
 1:      program mon2yearcmap
 2: ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 3:      implicit none
 4: c parameter
 5:      integer         nx, ny
 6:      parameter       (nx=144, ny=72)
 7: c variable
 8:      integer         ix, iy
 9:      integer         iyear, iyearmin, iyearmax, imon
10: c temporary
11:      character*128   ctmp
12: c function
13:      integer         iargc           !! Default function
14:      integer         igetday         !! Original function
15:      integer         len_trim        !! Original function
16: c in
17:      real            rmondat(nx,ny)  !! Monthly data array
18: c out
19:      real            ranudat(nx,ny)  !! Annual data array
20: c local
21:      integer         inofdaymon      !! Number of days in a month
22:      integer         inofdayyear     !! Number of days in a year
23: c files
24:      character*128   cdir, cexp, crun, csuf
25:      character*128   cifname, cofname
26:      integer         ilen
27:      character*128   clen
```

まず、第１ブロックでは変数の宣言をしています[31]。１３行目から２２行目までの「!!」以降の文字列はコメント文です。文章の途中からコメントを入れるこのやり方は標準的な FORTRAN の文法にはありませんが、たいていどのコンパイラでも使えるようで

[31] 変数宣言については第５章を参照のこと。

す。便利なので積極的に使い、コードを分かりやすくしましょう。

```
 1: ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 2: c Get Arguments
 3: ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 4:       if(iargc().ne.6)then
 5:         write(*,*) 'mon2year dir exp run yearmin yearmax suf'
 6:         stop
 7:       end if
 8: c
 9:       call getarg(1,cdir)
10:       call getarg(2,corg)
11:       call getarg(3,crun)
12:       call getarg(4,ctmp)
13:       read(ctmp,*) iyearmin
14:       call getarg(4,ctmp)
15:       read(ctmp,*) iyearmax
16:       call getarg(5,csuf)
17: c
18:       ilen=len_trim(cdir)
19:       write(clen,*) ilen
```

第２ブロックでは引数を読み込んでいます。このプログラムは６つの引数が必要です。引数を代入する変数ですが、ディレクトリ名、プロジェクト名、実験名、拡張子を示す変数 cdir, corg, crun, csuf は文字列型なのですが、開始年と終了年を示す変数 iyearmin, iyearmax は整数型です。FORTRAN の文法の制約上、引数には文字列型しかとれません。そこで、開始年と終了年はいったん call getarg を使って ctmp に文字列として読み込んだ後、read 文を使って ctmp という文字列型変数を整数型変数の iyearmin と iyearmax に読み込むという作業を行っています（13 行目と 15 行目）。read 文や write 文の基本的な使い方は第５章で学んだように、write(装置番号, 書式)あるいは read(装置番号, 書式)と使うのでしたね[32]。ここでは、装置番号に変数名が書かれていますが、これは変数を読んだり、書いたりすることを示します。

１８行目と１９行目では変数 cdir に代入された値の文字数を数えています。これはこのプログラムの後半で利用します。１８行目では len_trim という１次元の配列を変数 ilen に代入しているように見えますが、これは len_trim という関数(function)に cdir という引数を与えていることを示します。関数とは、コマンドの中におくコマンドだと考えてください。関数はこの例のように、一つの値（整数、実数、文字列）のようにふるまいます。この関数は引数の文字数を計算するもので、ソースコードは~/unix_semi/src/len_trim.f にあります。例えば「cdir='/home/naota'」の場合、len_trim(cdir)=11 となります。１９行目では整数型変数 ilen を文字列型変数 clen に書き込むことにより、変数の型を変換しています。先ほどの例の場合、整数 11（じゅういち）が文字列 11（いちいち）に変換されることになります。

[32] Read 文の場合、装置番号は「何から読むか」を示す。15 なら装置番号 15 が割り当てられているファイルから読むことを示す。Write 文の場合、装置番号は「何に書くか」を示す。16 なら装置番号 16 が割り当てられているファイルに書き出すことを示す。

```
 1: ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 2: c Sum up
 3: ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 4:       do iyear=iyearmin,iyearmax
 5: c clear variable
 6:         inofdayyear=0
 7: c clear array
 8:         do iy=1,ny
 9:           do ix=1,nx
10:             ranudat(ix,iy)=0.0
11:           end do
12:         end do
13: c
14:         do imon=1,12
15:           inofdaymon=igetday(iyear,imon)
16:           inofdayyear=inofdayyear+inofdaymon
17: c read data
18:           write(cifname,'(a'//clen//',a4,a4,i4.4,i2.2,i2.2,a5)')
19:      $    cdir,corg,crun,iyear,imon,0,csuf
20:          call read_binmat(rmondat,nx,ny,15,cifname)
21: c calclulation
22:           do iy=1,ny
23:             do ix=1,nx
24:               ranudat(ix,iy)
25:      $         =ranudat(ix,iy)+rmondat(ix,iy)*real(inofdaymon)
26:             end do
27:           end do
28:         end do
```

← 配列と変数に入っている値を消去する

第３ブロックと第４ブロックでは、do ループを使って繰り返し計算をしています。４行目の「do iyear=iyearmin,iyearmax」に対応する「end do」は第４ブロックの 12 行目にあります。つまり、この間の処理を繰り返すことになります。

最初の破線で囲まれた部分は後で説明することにして、14 行目の do imon = 1, 12 の行を見てください。ここでは、変数 iyear を値 iyearmin から値 iyearmax までループさせる最中に、さらに変数 imon を値 1 から値 12 までループさせています。これによって、iyearmin 年 1 月から iyearmax 年 12 月まで順々に処理を行うことができます。

15 行目は igetday という２次元の配列を読んでいるように見えますが、これは igetday という関数(function)に iyear と imon という２つの引数を与えて呼んでいることを示します。この関数は西暦○年○月が何日あるかを計算するもので、ソースコードは ~/unix_semi/src/igetday.f にあります。この関数はうるう年の２月の処理をきちんとこなすので、例えば igetday(1988,2)なら 29 となります。

18 行目から 19 行目にかけては write 文がありますね。この write 文の装置番号は文字列型変数 cifname となっています。これは結果を cifname に書き込むことを示すのでしたね。続いて write 文の書式ですが、これまでは自動設定を表す「*」を使ってきましたが、ここでは手動設定がされています。まず、シングルクオーテーションの内側にある括弧で囲まれた文字列が書式を示しています。つまり、「a'//clen//',a4,a4,i4.4,i2.2,i2.2,a5」が書式を表しています。結論から言うとこれは「clen 文字の文字列(a'//clen//')、4 文字の文字列(a4)、4 文字の文字列(a4)、4 桁の整数(i4.4)、2 桁の整数(i2.2)、2 桁の整数(i2.2)、5 文字の文字列(a5)を書き出す」ということを表します。まず、最初にある「a'//clen//'」は非常に変わった書き方ですが、「長さが clen の文字列」を表します[33]。例えば clen が 4 なら 4 文字、8

[33] //は文字列と文字列をつなぐことを意味する記号です。書式は文字列で記述する必要があるため、整数型変数 ilen ではなく文字列型変数 clen が使われています。

なら 8 文字となります。「a4」は文字列(ascii)が 4 つ、「i4.4」とあるのは 4 桁の整数で必ず 4 桁の整数で埋めること[34]を示します。「a4,a4」のように間にあるカンマ(,)は見やすさのためにあるので無視されます。このように write 文を使って書式を持った文字列を変数に代入しています（この場合、変数 cifname に CMAP203_19870100.cmap などと代入している）。

20 行目は call 文の後に read_binmat という 5 次元の配列が書かれているように見えますが、これは 5 つの引数を持つ read_binmat というサブルーチンを呼び出すことを意味します。サブルーチンとは、コマンドの中におくコマンドだと考えてください。このサブルーチンは「nx×ny の配列の cifname というファイル名のバイナリファイルを装置番号 15 で読み込み、結果を rmondat に代入する」機能を持ちます。詳細については次の節で説明します。

24 行目から 25 行目は変数 ranudat に各月のデータを足し込む作業をしています。

28 行目には do imon = 1, 12 に対応する end do があり、ループが終了します。

最後に、説明を飛ばしていた一つめの破線で囲まれた箇所の解説をしましょう。do ループを使って繰り返し処理を行ったとき、配列と変数には前の値が入っています。例えば 1987 年から 1988 年までの計算を行うとき、まず変数 ranudat には 1987 年の合計値が入りますが、何もしなければ 1988 年になっても残ってしまい、1987 年と 1988 年の合計を足すような処理になってしまうのです。そこで、各年の最初の足し込みを行う前に、配列と変数に入っている値を消去します。これは数字を足し合わせていくような処理では、とても重要な作業です。

```
 1: c Get average
 2:       write(*,*) 'Total num of days:', ndayyear
 3:       do iy=1,ny
 4:         do ix=1,nx
 5:           ranudat(ix,iy)=ranudat(ix,iy)/ndayyear
 6:         end do
 7:       end do
 8: c Write results
 9:       write(cofname,'(a'//clen//',a4,a4,i4.4,i2.2,i2.2,a5)')
10:     &  corg,crun,iyear,0,0,csuf
11:       call wrte_binmat(ranudat,nx,ny,16,cofname)
12:      end do
13: c
14:      end
```

第 4 ブロックでは、2 行目から 7 行目までで配列の平均値を求めています。このコマンドの処理を図で表すと、図 7-1 のようになります。その後 9 行目から 11 行目までで結果を書き出しています。ここでは、「read_binmat」とちょうど逆の操作をする「wrte_binmat」というサブルーチンを呼んでいます。このサブルーチンは「cofname という名前のバイナリファイルを装置番号 16 で開き、nx×ny の配列 ranudat を書き込む」という機能を持ちます。詳細については次の節で説明します。

[34]例えば 25 なら 0025 とするということです。

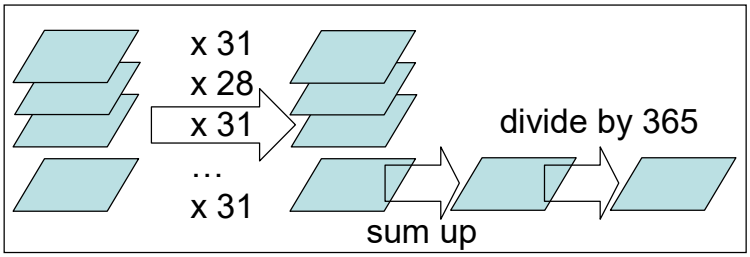


図 7-1 mon2yearcmap の計算手順

7.2 FORTRAN の Subroutine

mon2yearcmap.f では read_binmat と wrte_binmat の２つのサブルーチンが用いられていました。この節では、サブルーチンについて解説します。

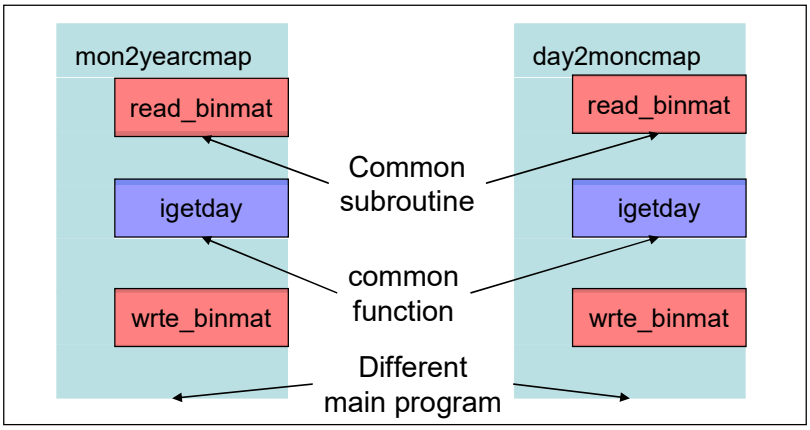


図 7-2 プログラム、サブルーチン、関数の関係

mon2yearcmap は月平均データを読み込み、年平均データを出力するコマンドですが、同様に日平均データを読み込み、月平均データを出力する day2moncmap というコマンドを作りたいとします。この２つのコマンドでは、ファイルを読み込んだり、書き出したりするタイミングは異なりますが、ファイルを読み込んだり、書き出したり、ある月の日数を計算したりする作業は共通して発生します。このとき、図 7-2 のように、mon2yearcmap と day2moncmap という別の主プログラムで、共通のサブルーチンと関数を使うことができます。このようにいくつかのプログラムで同じ処理を行いたい時にサブルーチンは効果的です。

まず、プログラムでサブルーチンを呼ぶ時には、

```
call サブルーチン名（引数１，引数２，引数３，...）
```

という形を使います。引数はサブルーチンに受け渡され、サブルーチンの中の処理が終わるとプログラムに返されます。

ではサブルーチン read_binmat.f のソースコードを見てみましょう。

```
 1: subroutine read_binmat(rdat,nx,ny,infile,cfname)
 2: ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 3:       open(infile,file=cfname,access='DIRECT',status='old',recl=4*nx)
 4:       do j = 1,ny
 5:         read (infile,rec=j)(rdat(i,j),i=1,nx)
 6:       end do
 7:       close (infile)
 8: c
 9:       end
```

まず１行目で、サブルーチンであることを宣言しています。かっこの中は引数で、プログラムから受け渡されます。引数は、書いてある順番通りに受け渡されるので、プログラムとサブルーチンで変数の名前が違っていてもかまいませんが、変数の型や配列の大きさは厳密に一致していなければいけません。たとえば mon2yearcmap の場合、プログラムの変数 rmondat の値がサブルーチンの変数 rdat に受け渡されることになります。その他の３行目から７行目までは第５章で学んだ構文と全く同じであることが分かると思います。このように「ファイルを開いて中に記述された数値を配列に代入する」という機能を抜き出したものをサブルーチンと呼びます。

7.3 make と Makefile

mon2yearcmap のようにプログラムとサブルーチン、関数が別々のファイルに用意されている場合、ひとつのプログラムに統合する作業が必要です。これを管理するためのプログラムが make コマンド、make コマンドに指示を与えるファイルが Makefile です。

mon2yearcmap をコンパイルするときにはまず mon2yearcmap.f というプログラムのソースコードと、read_binmat.f wrte_binmat.f というサブルーチンのソースコード、igetday.f, len_trim.f という関数のソースコードをそれぞれ FORTRAN コンパイラでコンパイルし、オブジェクトファイルという機械語に翻訳します。これらのオブジェクトファイルはそれぞれ、mon2yearcmap.o、read_binmat.o、wrte_binamt.o、igetday.o、len_trim..o という名前になります。次にこれらの５つのオブジェクトファイルを組み合わせ、mon2yearcmap という一つのコマンドにします。

mon2yearcmap の Makefile は~/unix_semi/src/にあります。この Makefile は mon2yearcmap をコンパイルする以外にも様々な設定が記述されているのですが、説明を簡単にするため、以下では mon2yearcmap に関するものだけ説明します。なお、このテキストでは Makefile の細かい文法について説明できませんので、自習して下さい。

```
 1:#################################################################
 2:#    Macro
 3:#################################################################
 4:RM     = /bin/rm        # path to command rm
 5:FC     = gfortran       # path to fortran compiler
 6:FFLAGS =                # option for fortran compiler
 7:#################################################################
 8:#    Suffix rule
 9:#################################################################
10:.f.o:
11:	$(FC) $(FFLAGS) -c $<
12:#################################################################
13:#    Dependency
14:#################################################################
15:TARGET1     = mon2yearcmap
16:COMPONENT1 = read_binmat.o wrte_binmat.o \
17:              igetday.o len_trim.o
18:#################################################################
19:#    Compilation
20:#################################################################
21:$(TARGET1) : $(TARGET1).o $(COMPONENT1)
22:	$(FC) -o $@ $@.o $(COMPONENT1)
```

Makefile は make の文法に沿って記述されたスクリプトです。この文法は様々な点で Bourne シェルスクリプトと異なりますが、構造は似ています。

１行目から３行目まではコメント文です。Makefile でも「＃」以降はコメントです。４行目から６行目は変数に値を代入しています。Bourne シェルスクリプトの場合は「=」の前後にスペースを入れてはいけないのですが、Makefile の場合は入れてもかまいません。16 行目のように途中で改行するときは、行末に\（バックスラッシュ）をつけ直後に改行を入れます。

１０行目から１１行目にかけては拡張子「.o」のオブジェクトファイルをどのように作るかが示されています。文法については説明しませんが、拡張子「.f」の FORTRAN ソースコードを変数 FC に代入された FORTRAN コンパイラを使ってコンパイルすることによってつくる、ということが示されています。

１５行目から１７行目にかけては、ファイルの依存関係についての変数を定義し、その値を代入しています。今作ろうとしているファイルは実行ファイル mon2yearcmap で、これを作るには、自分自身のオブジェクトファイル mon2yearcmap.o 以外に、read_binmat.o、wrte_binmat.o、igetday.o、len_trim.o の４つのオブジェクトファイルが必要です。このことを mon2yearcmap はこれら４つのオブジェクトファイルに依存しているといいます。この例では、作ろうとしているコマンドを代入する変数を TARGET1、依存しているオブジェクトファイルを代入する変数を COMPONENT1 としています。

２１行目から２２行目にかけては、コンパイルの命令をしています。まず２１行目はファイルの依存関係を示します。変数に値を代入すると次のようになります。

mon2yearcmap : mon2yearcmap.o　read_binmat.o　wrte_binmat.o　igetday.o　len_trim.o

「:」（コロン）記号は、その左側にあるファイルがその右側にあるファイルに依存していることを示します。つまり、右側にあるファイルのうち、左側にあるファイルより新しいものはコンパイルをやり直すことになります。２２行目はコンパイルの仕方が示されています。変数に値を代入すると次のようになります。

gfortran –o mon2yearcmap mon2yearcmap.o read_binmat.o wrte_binmat.o igetday.o len_trim.o

最初の「gfortran -o mon2yearcmap」は第５章で学んだように、実行ファイルを「a.out」と

いう名前ではなく「mon2yearcmap」という名前にすることを示すのでしたね。そのあとにオブジェクトファイル名が引数として続いていますが、これらは FORTRAN コンパイラによって[35]一つの実行ファイルに結合されます。

なお、破線で囲まれた部分は<tab>です。<space>ではエラーになるので注意してください。Make コマンドが実行されると同じディレクトリにある Makefile を探しますので、別のディレクトリにある Makefile は無視されます。

Make ファイルの実行方法

```
% make
```

ターゲットを指定して実行するには、

```
% make file
```

宿題

年と月を入力するとその月の日数を出力するプログラムを組みましょう。ファイルの名前は「program」としてください。コンパイルして、実行してみましょう。次のようになったでしょうか。

```
% program 1995 3
31
% program 1995 2
28
% program 1996 2
29
% program 2000 2
29
% program 1900 2
28
```

※宿題の解答例は~/unix_semi/lesson7 にあります。

[35] 正確に言うと、FORTRAN コンパイラがリンカという特別な UNIX コマンドを自動的に使ってオブジェクトファイルを結合します。

第 8 章

Bourne シェルスクリプト(2) Do ループ、For ループと If 文

第6章の宿題2では、最低 42 個のコマンドを端末に打ち込む必要がありました。同じようなコマンドを何度も入力するのは手間がかかる上、間違いをしやすいので避けたいところです。このような作業を減らすためにはどうすればよいのでしょうか。

8.1 シェルスクリプトの制御文と条件文

第 1 章や第 3 章で説明したように、UNIX ではテキストファイルにコマンドを保存し、実行権限を与えると、処理を実行できるのでしたね。この時に Bourne シェル文法に沿って記述されたファイルを Bourne シェルスクリプトといいます。Bourne シェル文法にはループや if 文などの制御文があり、これらを活用するとより高度な処理ができます。Bourne シェルスクリプトで使える主な制御文を表 8-1 に、条件文を表 8-2 にまとめました。

表 8-1 Bourne シェルスクリプトで使える主な制御文

For ループ	**for VAR in VALUES; do** **process** **done**
While ループ	**while [条件文]; do** **process** **done**

IF 文	**if [条件文]; then**
	process1
	elif [条件文]; then
	process2
	else
	process3
	fi

VAR: 変数名 VALUES: 値のリスト [条件文]:条件文 process: コマンド	

表 8-2 Bourne シェルスクリプトで使える条件文

FILENAME という ファイルがあるかの判定	**[-f FILENAME]** [36]
DIRNAME という ディレクトリがあるかの判定	**[-d DIRNAME]**
A＝B（文字列を判定する場合）	**[A = B]**

[36] [や]の前後には必ずスペースが必要です。

A≠B（文字列を判定する場合）	**[A != B]**
A＝B（整数を判定する場合）	**[A –eq B]**
A≠B 整数を判定する場合	**[A –ne B]**
A > B	**[A –gt B]**
A ≥ B	**[A –ge B]**
A < B	**[A –lt B]**
A ≤ B	**[A –le B]**
And	**-a**
Or	**-o**

8.2 【例1】For ループを使った Bourne シェルスクリプト

１月から 12 月までをあらわす１から１２までの数字を端末に表示し、誕生月（８月だとします）の場合に「Happy birth month!」と出力する Bourne シェルスクリプトを書いてみましょう。

```
 1: #!/bin/sh
 2: ##################################################
 3: MONS="01 02 03 04 05 06 07 08 09 10 11 12"
 4: BIRTHMON="08"
 5: ##################################################
 6: for MON in $MONS; do
 7:   if [ $MON = $BIRTHMON ]; then
 8:     echo $MON "Happy birth month!"
 9:   else
10:     echo $MON
11:   fi
12: done
```

１行目の#!/bin/sh は Bourne シェルスクリプトを宣言しています。３行目と４行目は、「MONS」「BIRTHMON」という変数それぞれに値を代入しています。６行目は「for ループ」という繰り返しの制御文です。ここにある「for MON in $MONS; do」は「変数 MON にリスト MONS にある値を順番に代入して、do と done の間にあるコマンドを繰り返す」ことを示します。この場合では MON=01, MON=02,… MON=12 と MON に MONS にある 12 の値を次々に代入することになります。７行目は「if 文」というは条件判定に基づく処理を行います。ここにある「if [$MON = $BIRTHMON]; then というのは、もし変数 MON が変数 BIRTHMON と一致するならば」という条件を表します。この条件を満たす場合は８行目が実行されます。「echo」は文字列を端末に出力するコマンドです。条件を満たさない場合は９行目にある「else」の次の行である 10 行目が実行されます。

8.3 【例2】While ループを使ったシェルスクリプト

１月１日から 12 月 31 日までを端末に表示し、誕生日（2 月 28 日だとします）には「Happy birth day !」と出力する Bourne シェルスクリプトを作りましょう。

```
 1: #!/bin/sh
 2: ##################################################
 3: YEAR=2005
 4: MONS="01 02 03 04 05 06 07 08 09 10 11 12"
 5: BIRTHMON="02"
 6: BIRTHDAY="28"
 7: ##################################################
 8: for MON in $MONS; do
 9:   DAY=1                        # refresh day
10:   DAYMAX=`nofday $YEAR $MON` # set the last day of the month
11:   while [ $DAY -le $DAYMAX ]; do
12:     if [ $MON = $BIRTHMON -a $DAY = $BIRTHDAY ]; then
13:       echo $MON $DAY "Happy birth day!"
14:     else
15:       echo $MON $DAY
16:     fi
17:     DAY=`expr $DAY + 1`      # day for next iteration
18:   done
19: done
```

この Bourne シェルスクリプトは例１を拡張して作られています。９行目から１８行目が大きく変わっていますね。８行目の For ループにより、変数 MON には 1 から１２までの値が代入されます。９行目では変数 DAY に 1 を代入しています。その後 10 行目では、○年○月の日数を調べるためのコマンド「nofday」を使っています（表 4-2）。これは第７章の宿題で皆さんが作ったものとも同じはずです。例えば「nofday 2005 01」の結果は 31 となります。ここで記号「`」（バッククオーテーション）は「バッククオーテーションで囲まれたコマンドの実行結果」を示します。つまり、YEAR=2005、MON=01 の場合、まずバッククオーテーションの中身が 31 と計算され、その結果が DAYMAX=31 というように変数に代入されるわけです。11 行目には「while ループ」という繰り返し処理があらわれます。ここで「while [$DAY –le $DAYMAX]; do」は「変数 DAY の値が変数 DAYMAX の値以下のとき、do から done までの間にあるコマンドを繰り返す」ことを示します。この do に対応する done は 18 行目にありますね。その直前の 17 行目に「DAY=`expr $DAY + 1`」という表現があるでしょう。expr コマンドは四則演算を行います。この場合、変数 DAY に１を足し、その結果を変数 DAY に代入する、という操作を表します。このとき、$DAY, +, 1 のそれぞれの間にスペースが必要ですから注意しましょう。

8.4【例3】第6章の宿題2を一つの Bourne シェルスクリプトで処理する

第 6 章の宿題２では、最低 42 回コマンドを端末に打ち込む必要がありました。この処理を一つのシェルスクリプトにまとめてみましょう。まずは第６章の宿題２で端末に打ち込んだコマンドをテキストファイルに書いてみましょう。なぜなら、このファイルに実行権限を与えるだけでシェルスクリプトになるからです。例えば、年間降水量が 0mm/year 以上 50mm/year 未満、50mm/year 以上 100mm/year 未満の地域に住む人口を求めるシェルスクリプトは次のようになるでしょう。

```
1: # 0mm/year =< Precipitation < 50mm/year
2: maskcmap pop.cmap            prc.cmap ge   0 temp.pop.ge0.cmap
3: maskcmap temp.pop.ge0.cmap   prc.cmap lt  50 temp.pop.ge0.lt50.cmap
4: sumcmap  temp.pop.ge0.lt50.cmap  0
5: # 50mm/year =< Precipitation < 100mm/year
6: maskcmap pop.cmap            prc.cmap ge  50 temp.pop.ge50.cmap
7: maskcmap temp.pop.ge50.cmap  prc.cmap lt 100 temp.pop.ge50.lt100.cmap
8: sumcmap  temp.pop.ge50.lt100.cmap 0
```

それではこのシェルスクリプトから共通の処理を見つけて、値を変数に置き換えてみましょう。たとえば、２行目から４行目と、６行目から８行目は上限値（50 か 100 か）と下限値（０か 50 か）が異なるだけで、作業は同じですね。そこで、上限値を変数 UL (upper limit)、下限値を変数 LL (lower limit)として、シェルスクリプトを書き換えてみましょう。また宿題２では人口と面積の２つについて処理をする必要がありましたから、pop を変数 DAT (data)としてやはり書き変えてみましょう。すると、次のように書き直せるはずです。こうすると、３行目から５行目と、８行目から１０行目は全く同じになりますね。

```
 1: # 0mm/year =< Precipitation < 50mm/year
 2: DAT=pop; LL=0; UL=50
 3: maskcmap ${DAT}.cmap       prc.cmap ge ${LL} temp.${DAT}.ge${LL}.cmap
 4: maskcmap temp.${DAT}.ge${LL}.cmap prc.cmap lt ${UL} temp.${DAT}.ge${LL}.lt${UL}.cmap
 5: sumcmap temp.${DAT}.ge${LL}.lt${UL}.cmap 0
 6: # 50mm/year =< Precipitation < 100mm/year
 7: DAT=pop; LL=50; UL=100
 8: maskcmap ${DAT}.cmap       prc.cmap ge ${LL} temp.${DAT}.ge${LL}.cmap
 9: maskcmap temp.${DAT}.ge${LL}.cmap prc.cmap lt ${UL} temp.${DAT}.ge${LL}.lt${UL}.cmap
10: sumcmap temp.${DAT}.ge${LL}.lt${UL}.cmap 0
```

次に、do ループと for ループ、if 文を組み合わせて、シェルスクリプトを完成させましょう。まず変数 JOB を定義し、LL=0, UL=50 を JOB=1、LL=50, UL=100 を JOB=2 としましょう。次に while ループを使って、この変数 JOB を 1 から 2 まで繰り返しましょう。同じように変数 DAT を定義し、for ループを使って pop から lndara まで繰り返しましょう。すると、以下のようになります。

```
 1: #!/bin/sh
 2: ################################################
 3: for DAT in pop lndara; do
 4:  JOB=1
 5:  while [ $JOB -le 2 ]; do
 6:   if  [ $JOB -eq 1 ]; then
 7:    LL=0; UL=50
 8:   elif [ $JOB -eq 2 ]; then
 9:    LL=50; UL=100
10:   fi
11:   maskcmap ${DAT}.cmap       prc.cmap ge ${LL} temp.${DAT}.ge${LL}.cmap
12:   maskcmap temp.${DAT}.ge${LL}.cmap prc.cmap lt ${UL} temp.${DAT}.ge${LL}.lt${UL}.cmap
13:   sumcmap temp.${DAT}.ge${LL}.lt${UL}.cmap 0
14:   JOB=`expr $JOB + 1`
15:  done
16: done
```

変数と制御文を利用してまとめることで、スクリプトをすっきりと書くことができましたね。

宿題

CMAP のサイトからダウンロードできるオリジナルデータ（アスキファイル）が~/unix_semi/dat_org にあります。第２章の宿題で利用した技術を使って、これらのアスキファイルを全て 144×72×4 byte の月別のバイナリファイルに変換するシェルスクリプトを書きましょう。このとき、バイナリファイルの名前が~/unix_semi/dat_bin にあるものと同じになるようにしましょう。

ヒント

桁を揃えて出力するには

```
% YR=2
% YR=`echo $YR | awk'{printf("%2.2d", $1)'`
```

というように入力しましょう。つまり、１行目で変数 YR に 2 という数字を代入し、２行目でいったんこれを echo コマンドで表示し、パイプを使って awk コマンドに渡し、awk の書式付プリントコマンド printf で２桁のうち２桁を必ず使う整数表記（2.2d）を指定します。詳しい文法が知りたい人は、インターネット検索して下さい。

また、西暦の下 2 桁を変数とした場合、使用するデータが 1999 年から 2000 年に変わるときにどのように対処すれば良いか考えましょう。例えば、1999 年までの処理と 2000 年以降の処理とを分けて 2 つのループ文を作ると良いでしょう。または、変数が 3 桁になったら変数から 100 をひいて 2 桁に戻すという方法も有効でしょう。

※宿題の解答例は~/unix_semi/lesson8 にあります。

Appendix A
Windows 機から UNIX サーバにログインする

Windows 機から UNIX サーバにログインするには PC X サーバというソフトウエアを利用する必要があります。この章で紹介するのは無償で提供されている Xming というソフトウエアです。このソフトをインストールし、基本的な使い方を覚えましょう。

この章では次のようなコンピュータ環境を前提にします。まず、UNIX あるいは Linux を含む UNIX 系のオペレーティングシステム（OS）が搭載されたコンピュータ（このテキストでは UNIX サーバと呼びます）がネットワークに接続されており、あなたのアカウントが作成されています。あなたは Windows OS が搭載されたコンピュータ（Windows 機と呼びます）を持っており、ネットワークに接続しています。これからあなたは Windows 機から UNIX サーバにログインし、UNIX 環境を利用します。あなたの持っている Windows 機をクライアント、UNIX サーバをサーバといいます。Windows コンピュータから UNIX コンピュータにログインして UNIX 環境を利用するには PC X サーバと総称されるソフトウエアが必要です。

A.1 Xming のインストール

1. Xming のサイト[37]に行きます。
2. “distributed”のリンクをクリックします。

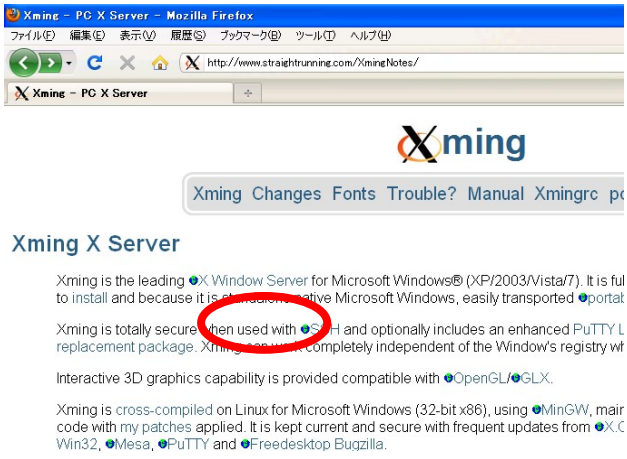


図 A-1 手順 2 のスクリーンショット

3. “SourceForge Project Xming”のリンクをクリックします。

[37] http://www.straightrunning.com/XmingNotes/

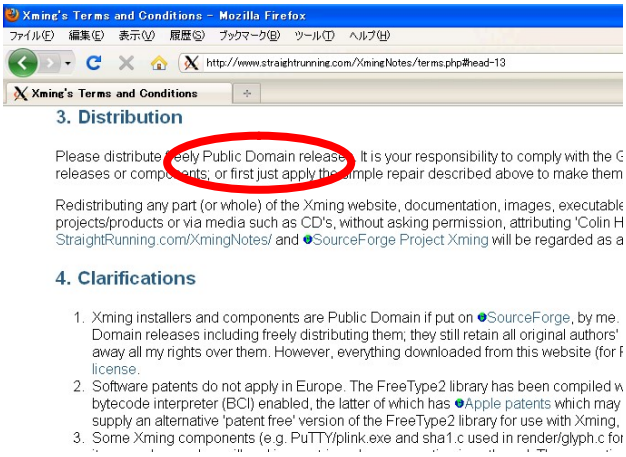


図 A-2 手順 3 のスクリーンショット

4. “View all files”のボタンをクリックします。

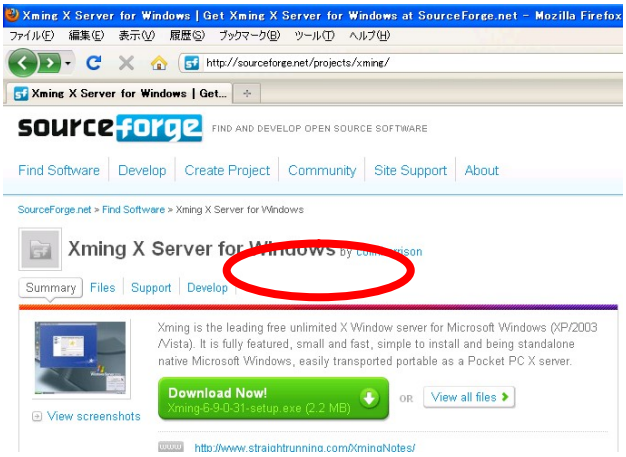


図 A-3 手順 4 のスクリーンショット

5. “Xming-fonts”をクリックし、現れた “7.5.0.11”（あるいは類似のもの）をクリックします。同様に、 “Xming”をクリックし、現れた “6.9.0.31”（あるいは類似のもの）をクリックします。“Xming-fonts-7-5-0-11-setup.exe”と“Xming-6-9-0-31-setup.exe”がブラウザに表示されていることを確認します。

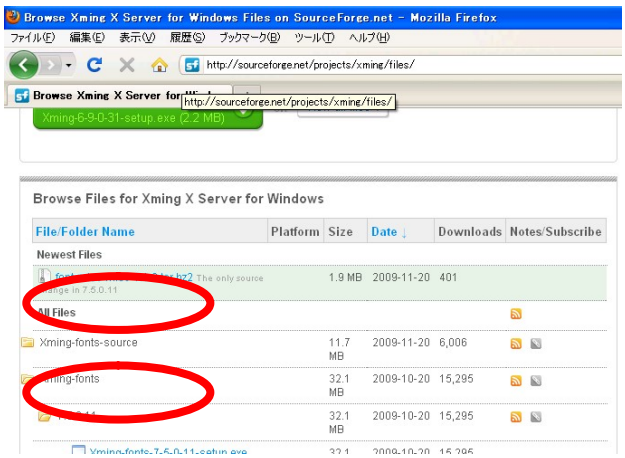


図 A-4 手順 5 のスクリーンショット

6. “Xming-fonts-7-5-0-11-setup.exe”と“Xming-6-9-0-31-setup.exe”がダウンロードされたことを確認します。
7. “Xming-fonts-7-5-0-11-setup.exe”を実行します[38]。基本的に、インストールが終了するまで「次へ」をクリックし続けて問題ありません。

図 A-5 手順 7 のスクリーンショット

8. “Xming-6-9-0-31-setup.exe”を実行します。基本的に、インストールが終了するまで「次へ」をクリックし続けて問題ありません。

図 A-6 手順 8 のスクリーンショット

A.2 Xming の使い方

1. Xming を起動します。スタートメニューから “Xlaunch”アイコンを探してクリックします。

[38] 重要：“Xming-fonts-7-5-0-11-setup.exe”からインストールするとスムーズ。もし、“Xming-6-9-0-31-setup.exe”からインストールした場合は、インストール後、必ず Xming を一度終了してから“Xming-fonts-7-5-0-11-setup.exe”をインストールすること。

図 A-7 使い方の手順 1 のスクリーンショット

2. “Multiple windows”を選択します。

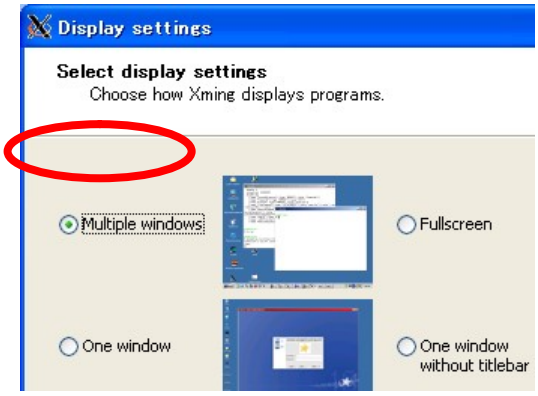


図 A-8 使い方の手順 2 のスクリーンショット

3. “Start a program”を選択して次へをクリックします。

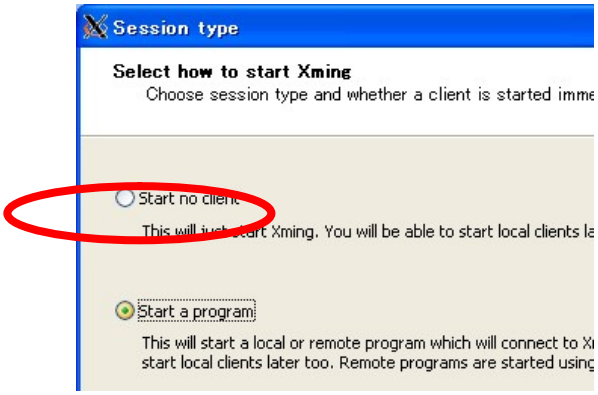


図 A-9 使い方の手順3のスクリーンショット

4. “Using PuTTY”を選択肢、”Connect to computer”の欄に接続する UNIX コンピュータ

名を入力します。後の２つの欄は空欄のままにします。

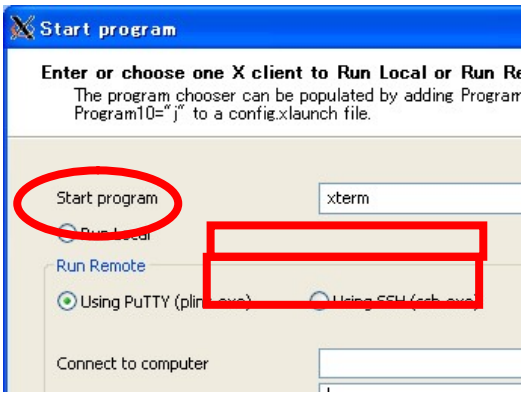


図 A-10 使い方の手順4のスクリーンショット

5. もう１枚ウィンドウがあらわれるが、何も選択せずに次へをクリックすます。

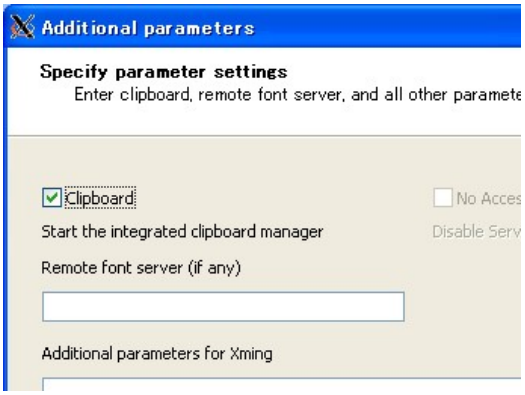


図 A-11 使い方の手順5のスクリーンショット

6. この選択を保存したいときは “Save configuration”をクリックする。次回以降、手順 1～6 を省略できる。最後に完了をクリックします。

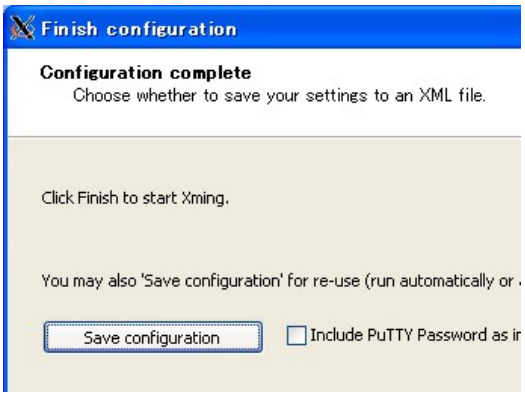


図 A-12 使い方の手順6のスクリーンショット

7. 最初に接続するときには次のようなウィンドウがあらわれます。はいを選択します。

図 A-13 使い方の手順 7 のスクリーンショット

8. ID（ログイン名）を入力します。

図 A-14 使い方の手順 8 のスクリーンショット

9. パスワードを入力します。

図 A-15 使い方の手順9のスクリーンショット

10. このような画面が現れたら接続に成功です。

図 A-16 使い方の手順 10 のスクリーンショット

A.3 困った時に

うまくいかないときは一度 Xming を終了し、XLaunch を立ち上げるところから再開してください。 このとき、タスクバーにある X のアイコンを探し、右クリックして終了(Exit)してください。

図 A-17 Xming の Exit の仕方

うまくログインができないときは、ログ（履歴）を見るとヒントが見つかるときがあります。同じくタスクバーにある X のアイコンを右クリックして View Log をクリックして下さい。

図 A-18 Xming の Log の見方

A.4 必要なソフトウエアの導入

このテキストを実施するのに必要となる UNIX 環境を表 A-1 に示します。これらの UNIX 環境を設定するには基本的に管理者権限が必要です。管理者の方に頼んで UNIX 環境を設定してもらうようにして下さい。

ログインシェル	任意ですが、このテキストでは bash を前提に説明をしています。
通信	ssh によるリモートログインが有効であること。 また X11Forwarding が有効であること。
ソフトウエア	以下のソフトウエアが利用可能なこと make gcc（GNU C コンパイラ） GMT（Generic Mapping Tool） imagemagick gfortran（GNU FORTRAN コンパイラ）

Appendix B

Mac を UNIX 環境として利用する

Apple 社の販売するコンピュータに搭載されている OS X は UNIX ベースです。つまり、Apple 社のコンピュータを UNIX 環境として利用することができます。

B.1 端末

XQuartz という無料のソフトウエアがインターネット上で配布されているので、インストールしてください。これを起動すると UNIX 端末として利用できます。

B.2 UNIX 環境構築の事例(Mac OS 10.6 の場合)

以下は、2009 年 11 月に筆者が Mac mini Server (Mac OS 10.6 Snow Leopard Server)に A.4 節に示した環境を設定した際の手順です。ご参考までに。

1. 初期設定
   - 初期設定のままでは Command-Space というキーボード操作に Spotlight の起動が割り当てられている。これは emacs 使用時に困るので、「システム環境設定」にある「キーボード・マウス」のアイコンの「ショートカット」タブで Spotlight のショートカット割当を変更する。
   - バックスラッシュ(\)が入力できないときは、Option(Alt)-¥と打つと出力される。
2. make と gcc のインストール
   - 初期設定のままでは make や gcc が入っていないので、Apple のサイト[39]から、まず Xcode をダウンロードする。ダウンロードするには ADC membership が必要。ダウンロードしたら、Xcode.mpkg をダブルクリックしてインストール。これで make や gcc がインストールされる。
3. fink のインストール
   - fink は UNIX ベースのオープンソースプログラムを OS X 上にインストールし、管理するためのツール。
   - fink のサイト[40]に行く。インストール時には、設定項目が多数あるが、基本的には全てデフォルトでよい。
   - FinkCommander をインストール。
4. GMT と imagemagick のインストール
   - Fink を使ってインストールする。ただし、imagemagick は別の方法[41]の方がインストールにかかる時間が短い。
5. gfortran をインストール

[39] http://developer.apple.com/tools/xcode/
[40] http://www.finkproject.org/
[41] ImageMagick のサイト(http://www.imagemagick.org)では Mac OS X にインストールする際に、Fink とよく似た機能を持つ MacPorts からインストールすることを推奨している。そこで、まず MacPorts のサイト（http://www.macports.org/）に従って Package をインストール。次に指示に従って環境変数を設定する。

- gfortran のサイト[42]から Mac OS10.5 (Leopard)用のバイナリをダウンロード。これは Mac OS10.6 (Snow Leopard)でも問題なく動作する。

[42] http://gcc.gnu.org/fortran/

Appendix C

ubuntu とソフトウエアの導入

ubuntu は利用者の多い Linux ディストリビューションの一つです。この章では、ubuntu をコンピュータに導入する方法と、このテキストを実施するために必要となるソフトウエアのインストール方法について述べます。ubuntu の導入方法には、ubuntu を単一の OS とする場合と ubuntu と Windows の両方を OS とするデュアルブートの場合がありますが、この章では前者についてのみ記述します。ubuntu の ISO イメージファイルを入手し、インストール用の USB メモリを作成するには、別途 Windows が導入されたコンピュータが必要となります。

C.1 ubuntu 導入の準備

この節では、ubuntu 導入の準備について記述します。この準備は ubuntu を導入するコンピュータとは別の（あるいは ubuntu を導入する前で Windows が導入されている）コンピュータが必要です。準備のおおまかな流れは、ubuntu の ISO イメージファイルを入手し、Universal USB Installer を使って USB メモリに ubuntu を書き込みます。

（1）ubuntu ISO イメージファイルをダウンロードします。

1. ubuntu の HP にアクセスします。以下は ubuntu Japanese Team (http://www.ubuntulinux.jp/) の HP にアクセスした場合について示します。
2. ubuntu のダウンロードをクリックします。
3. 日本語 Remix イメージのダウンロードをクリックします。
4. 任意の ISO イメージファイルをダウンロードし、デスクトップなど分かりやすい場所に保存します。

（2）ubuntu の ISO イメージファイルを書き込んだ USB メモリを準備します。

1. 新品または不要な USB メモリ（1GB あれば十分）を用意し、PC に設置します。（使用済み USB メモリの場合は、事前にデータをバックアップしておくこと。）このとき、どのドライバーに USB メモリが刺さっているかを確認します。
2. Universal USB Installer の set up exe.をデスクトップなどに保存します。
3. Set up exe を実行します。
4. I Agree. をクリックします。

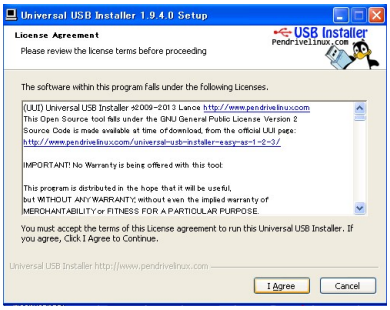


図 C-1 Universal USB Installer の利用誓約確認画面

5. （1）手順 4 で保存した ISO イメージファイルのバージョンを選択します。

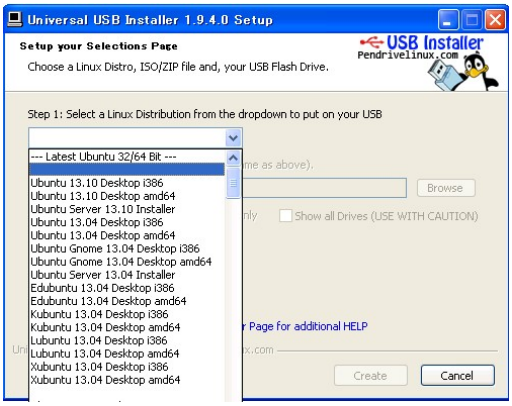


図 C-2 ubuntu のバージョンの選択画面

6. Browse をクリックし（1）手順 4 で保存したファイルを選択します。

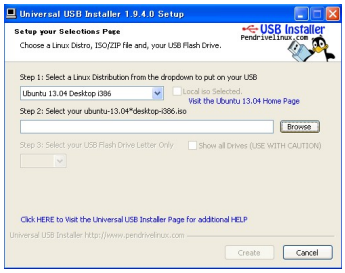


図 C-3

7. （2）手順 1 で設置した USB のドライバーを選択します。USB をフォーマットしたい場合は、チェックボックスにチェックします。

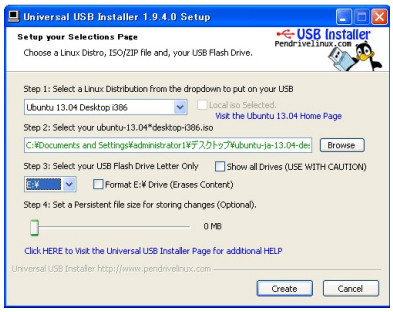


図 C-4

8. 使用する容量を設定する。（1）手順 4 で保存した ISO イメージファイルの容量より少し多くとります。

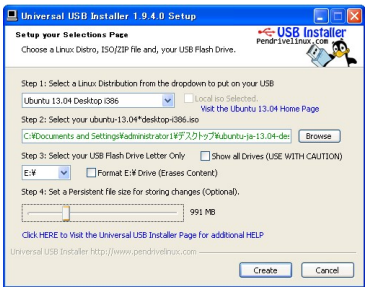


図 C-5

9. Create をクリックし実行します。

10. 以下のメッセージが表示されるのではいをクリックします。

図 C-6 Universal USB Installer の実行確認画面

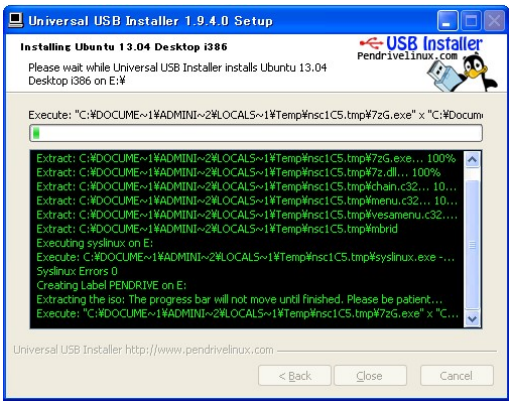


図 C-7 Universal USB Installer の実行画面

11. 終了したら Close をクリックします。

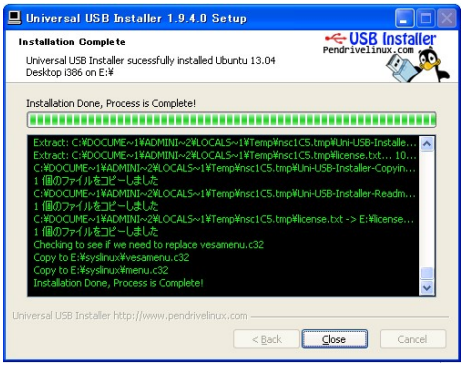


図 C-8 Universal USB Installer の実行終了画面

C.2 ubuntu の導入

この節では、ubuntu を導入する手順について述べます。

1. ubuntu を導入したいコンピュータに C.1（2）で用意した USB メモリを差し込んで、コンピュータを起動します。
2. Windows が起動する前にブートメニュー選択画面を表示します。（F12 などを押すと表示されます。）
3. USB メモリからブートするを選択します。
4. USB メモリに記録された ubuntu が起動します。
5. ubuntu をインストールを選択します。[43]
6. インターネットに接続し、続けるを選択します。インターネットに接続されてなくても特に問題はありません。
7. インストール方法の選択画面が表示されるので、ubuntu のみを OS とする方法を選択します。

   ※Windows とのデュアルブートを選択する場合は、よく注意して下さい。なお、Windows が消去されても一切の責任を負いません。
8. インストールが開始されます。その間、コンピュータの情報を設定します。
9. 今すぐ再起動するを選択します。
10. インターネットに接続します。
11. XTerm を起動し、以下のコマンドを入力してアプリケーションの状態を最新にします。

```
% sudo apt-get update
% sudo apt-get upgrade
```

C.3 必要なソフトウエアの導入

このテキストをを実施するのに必要となるソフトウエアは、emacs, imagemagick, fortran, GMT, netCDF の 4 つです。

まず、以下のコマンドを入力して emacs を導入します。

```
% sudo apt-get install emacs
```

次に、以下のコマンドを入力して imagemagick を導入します。

```
% sudo apt-get install imagemagick
```

次に、以下のコマンドを入力して gfortran を導入します。

```
% sudo apt-get install gfortran
```

[43] 参照：http://www.ubuntu.com/download/desktop

GMT と netCDF の導入方法は 2 通りあります。

1. 上記と同じように、apt-get コマンドを使用してインストールします。ただし、netCDF と gfortran の整合性に問題が発生することがあります。このテキストを実施するうえで問題はありませんが、netCDF のライブラリを含む fortran ソースコードがうまくコンパイルできない問題が発生します。

```
% sudo apt-get install gmt
% sudo apt-get install gmt-coast-low
% sudo apt-get install netcdf-bin
```

2. 少し手間がかかりますが、GMT のウェブサイトから GMT と netCDF をソースからインストールするスクリプトが配布されているのでそれを利用してインストールします。詳細は GMT のウェブサイト[44]をご覧ください。

[44] http://gmt.soest.hawaii.edu/

Appendix D

テキスト教材の導入と環境設定

この章では、本テキストの教材の導入方法と環境設定について記述します。この章に書いてあることを実施するのに管理者権限は必要ありませんが、基本的な UNIX 操作の知識が必要です。このテキストを使って初めて UNIX を勉強する方は管理者の方に頼んで、代わりに設定してもらうようにして下さい。

D.1 テキスト教材の導入と環境設定

このテキストを利用するユーザのホームディレクトリに移動し、テキストの教材をダウンロードしましょう。なお、各コマンドの役割については、本テキスト表 1-1 などを参考にして下さい。まず、ウェブブラウザを起動します。以下の例では、firefox がシステムにインストールされていることを仮定しています。

```
% cd ~
% firefox
```

1. http://h08.nies.go.jp/h08/index_j.html にアクセスします。
2. マニュアルに移動します。
3. 教材をダウンロードします。

次に、tar アーカイブを伸張しましょう。

```
% gunzip unix_semi.tar.gz
% tar xf unix_semi.tar
```

こうすると、unix_semi_20140301 のようなディレクトリができたはずです。ディレクトリ名が長いので次のようにして unix_semi という名前に変更しましょう。

```
% mv unix_semi_20140301 unix_semi
```

次に環境変数を設定します。ホームディレクトリにある.bashrc を編集します。

```
% emacs ~/.bashrc
```

まず、~/unix_semi/src にパスを通します。これによって src 内にあるサンプルコマンドがどこのディレクトリからも使えるようになります。

```
1:#################################################################
2:# PATH                                          ←コメント文：タイトル
3:#################################################################
4:PATH=.:${PATH}:~/unix_semi/src                  ←パスの設定
```

編集が終わったら、設定を有効にします。次のコマンドを実行して下さい。

```
% source ~/.bashrc
```

次にサンプルコマンドをコンパイルします。ディレクトリ src にはサンプルコマンドのプログラム文が保存されており、サンプルコマンドをコンパイルすることでそれらのコマンドが使用可能になります。ディレクトリ src にある Makefile の 10 行目と 12 行目が以下のようになっているか確認しましょう。

10 行目　FC = gfortran

12 行目　FL = gfortran

確認が済んだら、サンプルコマンドをコンパイルしましょう。

```
% cd ~/unix_semi/src
% make veryclean
% make all
```

最後に unix_semi で使用する気象データなどのアスキファイルをバイナリファイルに変換します。

```
% cd ~/unix_semi
% sh start.sh
```

これで準備完了です。

D.2 Intel Fortran Compiler を使う場合

もし gfortran でなく、Intel Fortran Compiler を利用する場合は次のようにして下さい。

1. write 文を使ったバイナリ出力で指定する recl 値の単位を byte に変えるため[45]、~/.bashrc に次のように加えます。
   alias ifort=’ifort –assume byterecl’
2. また~/unix_semi/src /Makefile を次のように書き変えます。
   10 行目 FC = ifort -assume byterecl
   12 行目 FL = ifort -assume byterecl
3. ~/unix_semi/src にあるコードを再コンパイルします。

```
% cd ~/unix_semi/src
% make veryclean
% make all
```

次に unix_semi で使用する気象データなどのアスキファイルをバイナリファイルに変換します。

[45] gfortran では write 文の recl 値の単位が byte ですが、Intel Fortran Compiler ではデフォルトで record となっています。このため、設定変更を行わないと real が 16 バイトで記録されることになり、計算値が異常になります。

```
% cd ~/unix_semi
% sh start.sh
```

これで準備完了です。

D.3 Big Endian で入出力する場合

以下に示すことはほとんどの方にとって不要ですので、読み飛ばして構いません。あなたがもし Big Endian でバイナリを記述したい場合は次のようにして下さい。その際、クォーテーションマークに注意してください。
~/.bashrc をエディタで開き、gfortran を使っている場合は、
export GFORTRAN_CONVERT_UNIT="big_endian;little_endian:25-26"
ifort を使っている場合は、
export F_UFMTENDIAN=" big;little:25-26"
という行を追加し、

```
% source ~/.bashrc
```

を実行してください。そして、~/unix_semi/src にあるプログラムを全て再コンパイルします。

```
% cd ~/unix_semi/src
% make veryclean
% make all
```

次に unix_semi で使用する気象データなどのアスキファイルをバイナリファイルに変換します。

```
% cd ~/unix_semi
% sh start.sh
```

これで準備完了です。

謝辞

第１版（英語の PowerPoint）の謝辞

このテキストは 2002 年から 2005 年までの夏学期に東京大学生産技術研究所沖研究室で行われた初心者向けの UNIX 講座の補助資料として作られました。プログラミングや UNIX の知識が全くない学生が沖研究室で研究を始めるために最低限必要な知識をまとめて紹介しています。
私の拙い UNIX 講座に参加してくれた右の学生さんに感謝します。山田朋人さん、須賀可人さん、A. Aslam さん、C. Manusthiparom さん、C. Apirumanekul さん、岡澤毅さん、小久保武さん、高垣佳永子さん、斎田渉さん、久保賢一さん、小岩祐樹さん、石崎安洋さん、Q. Tang さん、N.S. Farzin さん、J. Cho さん、M. Lin さん、犬塚俊之さん、咲村隆人さん、碇大輔さん、新井裕子さん、内海信人さん。
沖研究室は地球規模の水循環に関する研究を行っています。その際に膨大な時系列グリッドデータの操作が必要なため、このテキストはここに重点を置いた内容になっています。あえて内容を偏らせることで研究にすぐに役立つように工夫しました。
このテキストは<なんとか>英語で作りました。沖研究室に毎年やってくる熱意ある留学生に言葉のバリアをはらないためです。テキストの英語は文法・用法上の誤りが多々あるかもしれませんし、下手な英語で内容が分かりにくくなっているかもしれませんが、どうぞご勘弁ください。

2005 年 7 月 5 日
花崎直太

第２版（日本語の Word）の謝辞

これまで筆者の UNIX 講座の補助資料は英語で作成されていました。これは留学生の多い沖研究室で言葉のバリアを作らないための筆者なりの配慮でしたが、日本人の学生さんからはコンピュータ用語が英語で分かりにくいという苦情が寄せられていました。また、補助資料は筆者が講義をする際の Visual Aid として使いやすいように MS Powerpoint で作成されていました。このため、文章が少なく、復習したり独学したりするのに使いにくいという苦情もありました。そこで、すばらしい翻訳者・編集者の新田友子さんの協力を得て、2007 年６月から７月にかけて英語の MS Powerpoint スライドを日本語の文章によるテキストに再編集する作業を行いました。筆者が十分な時間をかけられなかったため、説明が不足していたり、論理が飛躍していたりするかもしれません。このテキストを読まれた方は、是非そうしたコメントを筆者までお寄せください。少しずつ改訂していきます。

2007 年 7 月 31 日
花崎直太