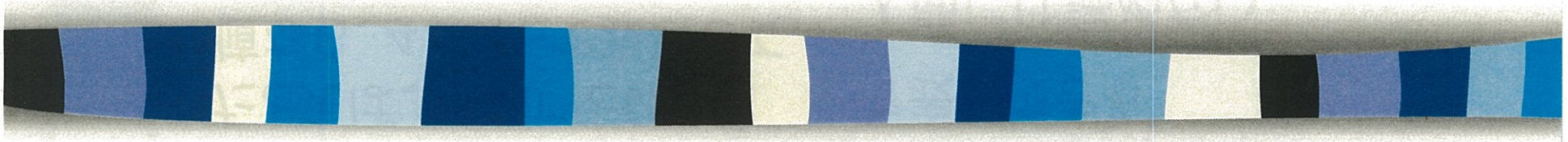
GrADS, GRIB & NetCDF Data



データの読み込みと、処理の方法

2002, 9/13 (Fri)

テキストデータとバイナリデータ

テキストデータ

アスキーコードや2バイト文字コードで構成されている

たとえば、“10.0” という数値なら、4バイトで表現している

エディタなどで開いてみることができる

データは大きくなる

バイナリデータ

16進数の羅列としてデータを構成する

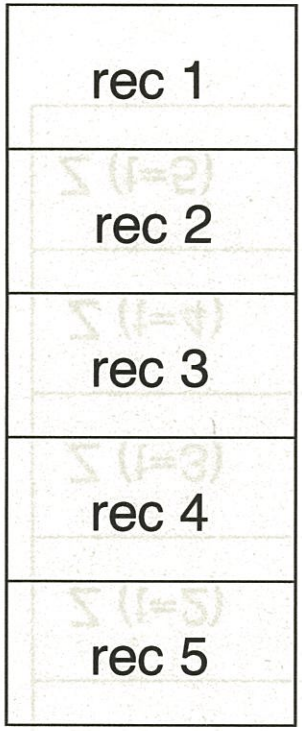
“10.0” なら 00A0 と2バイトで表現できる

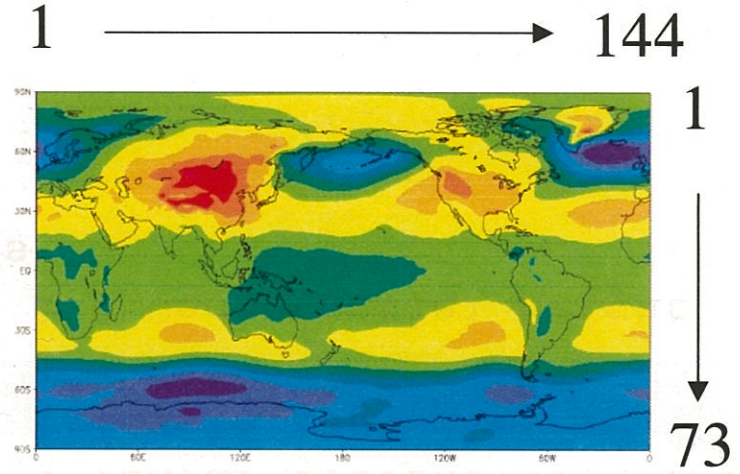
人間には読めない

データは小さくなる

Part1: GrADS binary

あだ名：Fortran binary, unformatted-direct, がっつん読み





144 x 73 = 10512 grid
1grid = 4 byte
→ 10512 x 4 = 42048bytes

バイナリデータのかたまり。

どのようにデータが入っているかは、ctl ファイルと一対一に対応している

```
dset ^slp.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
```

GrADS データの構造(1)

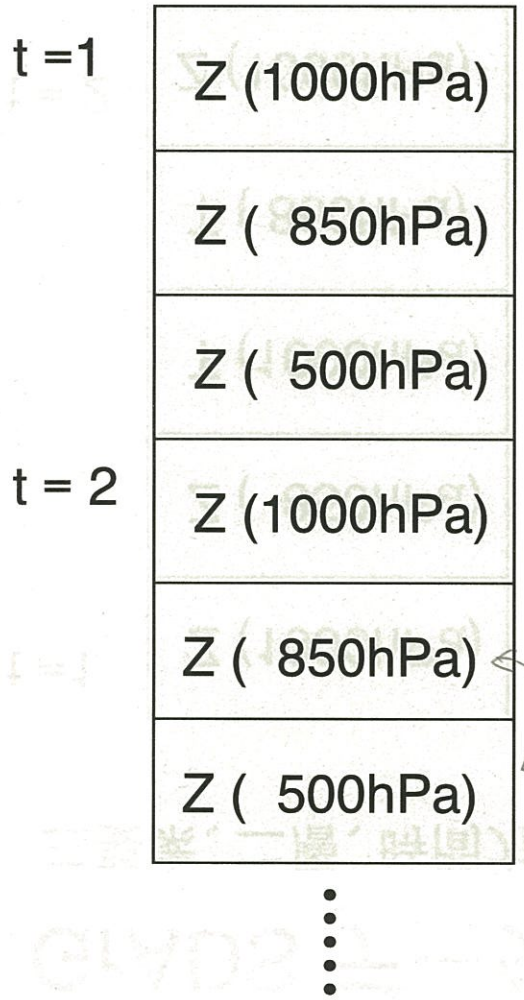
一要素、一層、時間方向にだけ複数のレコードがある

Z (t=1)
Z (t=2)
Z (t=3)
Z (t=4)
Z (t=5)

```
dset ^sst.mon.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
tdef   5 linear 00Z01nov1981 1mo
zdef   1 levels 1000
vars   1
z      1 0 Geopotential Height
endvars
```

GrADS データの構造(2)

一要素、三つの層、時間方向にも複数レコードがある



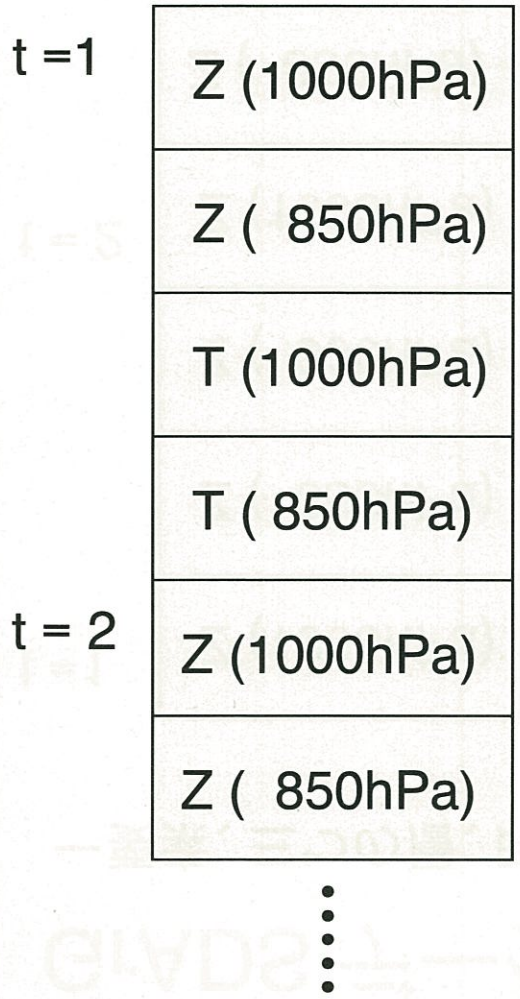

```
dset ^sst.mon.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
tdef   1 linear 00Z01nov1981 1mo
zdef   3 levels 1000 850 500
vars   1
slp    3 0 Sea Level Pressure
endvars
```

隣り合ったレコードは層が違う。

時間はもっともゆっくり変化する。

GrADS データの構造(3)

三要素、二層、時間方向にも複数レコードがある




```
dset ^sst.mon.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
tdef   2 linear 00Z01jan1981 1mo
zdef   2 levels 1000 850
vars   2
z    2 0 Geopotential Height
t    2 0 Temperature
endvars
```

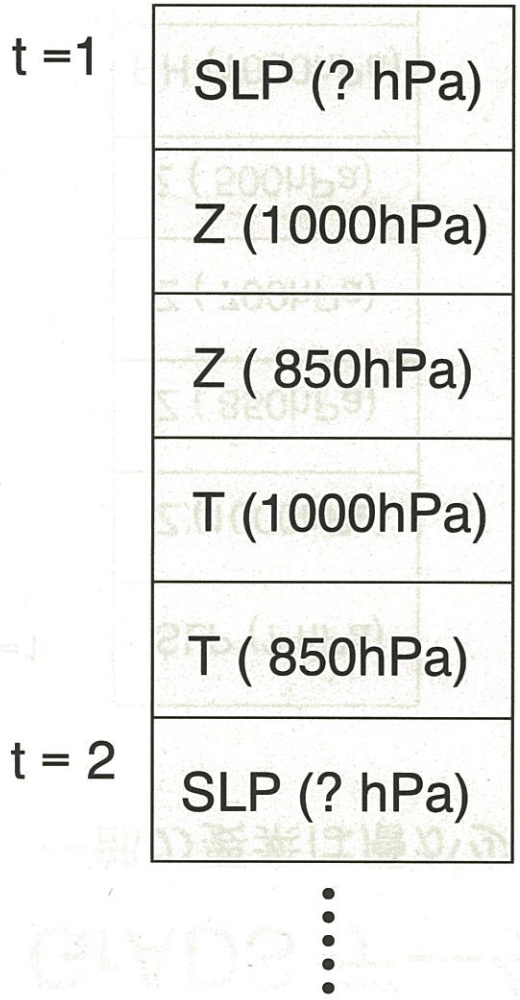
隣り合ったレコードは層が違う。

次に要素が変化する

時間はもっともゆっくり変化する。

GrADS データの構造(4)

表層の要素を含むデータ




```
dset ^sst.mon.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
tdef   2 linear 00Z01jan1981 1mo
zdef   2 levels 1000 850
vars   3
slp  0 0 Sea Level Pressure
z    2 0 Geopotential Height
t    2 0 Temperature
endvars
```

表層データは　0層のデータ

GrADS データの構造(5)

一部の要素は層が少ない場合

t =1

SLP (? hPa)
Z (1000hPa)
Z (850hPa)
Z (700hPa)
Z (500hPa)
RH (1000hPa)
RH(850hPa)
T (1000hPa)
T (850hPa)

```
dset ^sst.mon.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
tdef   2 linear 00Z01jan1981 1mo
zdef   4 levels 1000 850 700 500
vars   4
slp  0 0 Sea Level Pressure
z    4 0 Geopotential Height
rh   2 0 Relative Humidity
t    4 0 Temperature
endvars
```

層が少ない要素は、途中で
やめて次の要素にいく

GrADS データの読み書き (1)

1. 配列宣言

```
parameter(ilon=144, ilat=73)
real*4 data(ilon,ilat)
```

4バイトの実数を使う
配列は「経度、緯度、時間」の順にする

2. ファイルを開く

```
 open(10, file='slp.bin',
+form='unformatted',access='direct', recl=144*73*4)
```

これは魔法の呪文だと思って
そのままつかう。

GrADS データの読み書き (2)

3. 読み

```
read(10, rec=1) data
```

この一行で、data(ilon,ilat) にまるごと
データが読み込まれる

4. 典型的な処理

```
do i = 1,ilat
  do j = 1,ilon
    write(6,*) j,i, data(j,i)
  end do
end do
```

ループは配列の右側から、
「時間、緯度、経度」の順で内側にする

GrADS データの読み書き (3)

5. 書き

```
write(10, rec=1) data
```

この一行で、data(ilon,ilat) にまるごとデータが書き込まれる

Big / Little エンディアン

1バイトの情報では、00 ～ FF で 255という数字までしか表現できない。

それ以上の数字は、2バイト、4バイトといった大きさで格納する → `real*4` というのはこのこと

でも、0A 0B 0C 0D というバイトをどの順番で格納するかは、コンピューターによって違う！

Big Endian

“0A0B0C0D” と、桁の大きいほうを先に格納する

Motorola や Sun-SPARC のCPUのエンディアンはこれ

Little Endian

“0D0C0B0A” と、小さい桁を先に格納する

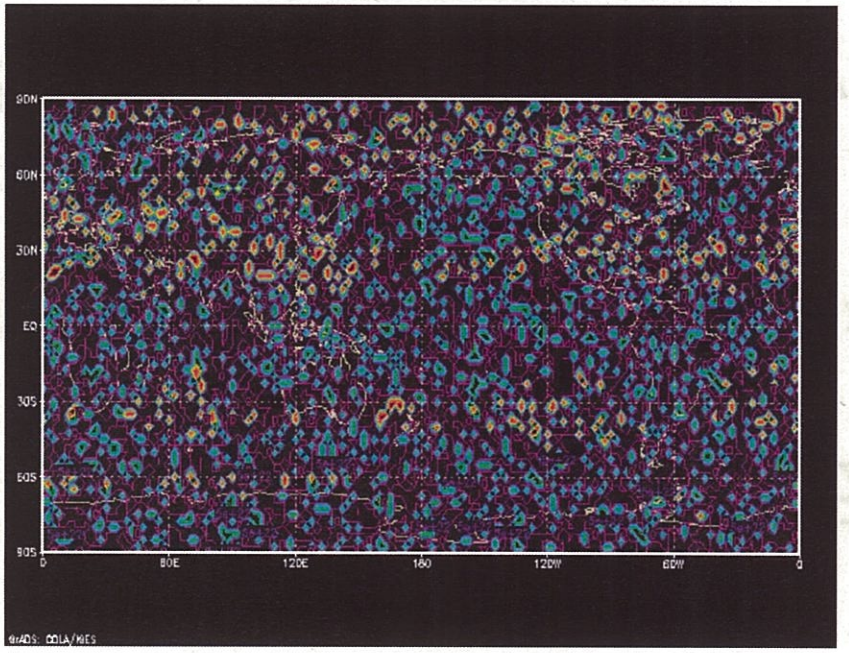
Intel や DEC-Alpha、PowerPC のCPUはこれ

Big Endian のデータを PC で読み込むと、

特徴的な、ばかでかい数字が読み込まれる。

4.60060299E-41	1だったはず
8.96831017E-44	2
2.30485571E-41	3
4.60074312E-41	4
5.74868682E-41	5

特徴的な、ぐちゃぐちゃの図になる



エンディアン問題の回避方法

データはだいたい Big Endian で統一されている

Fortran ではいまのところ Big で読み込んで Little で出力、といった便利なことができるコンパイラは少ない

→ PC上で作業する場合でも、Big で読み書きできる Fortran を使い、作業はすべて Big で統一する

Big Endian の入出力ができるもの

沼口さんパッチをあてた f2c

- f77 のかわりに使用するフィルタ
- f77 –ee というオプションで、big の入出力を行う

Fujitsu Fortran Compiler

- コンパイルしてから、実行ファイルに `-Wl,-T` というオプションをいれて使うと、big のI/O ができる

GrADS データの作り方：fwrite

データがすでにあって、その一部だけを出力したい場合は fwrite を使うほうが早い

ただし、出力されるデータのエンディアンはそのマシンのものに統一される （PCは注意！）

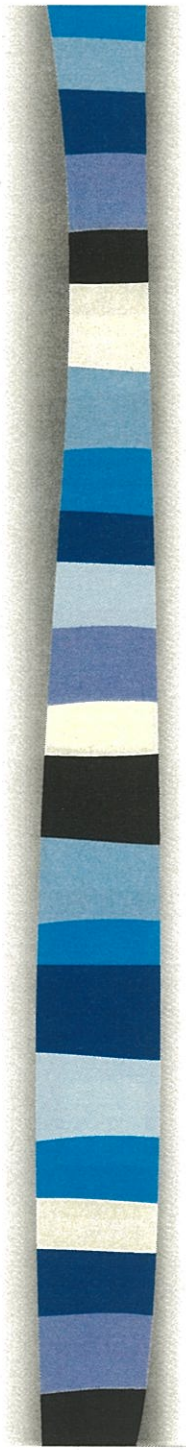
```
open slp.mon.mean.ctl

set x  1 144

set fwrite sample.bin

set gxout fwrite

d slp
```



Part2: GRIB

SDF (self-descripting-format) の一種で、WMO標準のデータ形式

NCEP や GPV データはこれで配信されている

読み込みはできるけど、書き込みは難しくて日常作業では非現実的

wgrib: GRIB 変換コマンド

ソースコードをダウンロードして、

```
cc -o wgrib wgrib.c -lm
```

でコンパイルして、/usr/local/binにでもインストールしておけばいつでも使える

wgrib の簡単な使い方

中身を見る

```
wgrib hoge.grib
```

```
1:0:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=1000:TR=113:P1=0:P2=6:TimeU=1:1000 mb:anl:ave@6hr:NAve=124
2:17166:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=925:TR=113:P1=0:P2=6:TimeU=1:925 mb:anl:ave@6hr:NAve=124
3:34332:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=850:TR=113:P1=0:P2=6:TimeU=1:850 mb:anl:ave@6hr:NAve=124
4:51498:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=700:TR=113:P1=0:P2=6:TimeU=1:700 mb:anl:ave@6hr:NAve=124
5:68664:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=600:TR=113:P1=0:P2=6:TimeU=1:600 mb:anl:ave@6hr:NAve=124
6:85830:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=500:TR=113:P1=0:P2=6:TimeU=1:500 mb:anl:ave@6hr:NAve=124
7:104310:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=400:TR=113:P1=0:P2=6:TimeU=1:400 mb:anl:ave@6hr:NAve=124
8:122790:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=300:TR=113:P1=0:P2=6:TimeU=1:300 mb:anl:ave@6hr:NAve=124
9:141270:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=250:TR=113:P1=0:P2=6:TimeU=1:250 mb:anl:ave@6hr:NAve=124
10:159750:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=200:TR=113:P1=0:P2=6:TimeU=1:200 mb:anl:ave@6hr:NAve=124
```

最初の3つのHGT だけを選ぶ

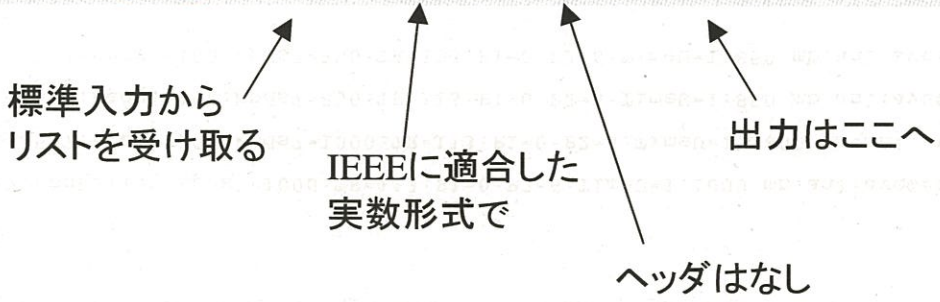
```
wgrib hoge.grib |  grep HGT  |  head -3
```

```
1:0:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=1000:TR=113:P1=0:P2=6:TimeU=1:1000 mb:anl:ave@6hr:NAve=124
2:17166:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=925:TR=113:P1=0:P2=6:TimeU=1:925 mb:anl:ave@6hr:NAve=124
3:34332:d=02010100:HGT:kpds5=7:kpds6=100:kpds7=850:TR=113:P1=0:P2=6:TimeU=1:850 mb:anl:ave@6hr:NAve=124
```

wgrib の簡単な使い方 (2)

GrADS データに変換

```
wgrib  hoge.grib  |  grep HGT  |  head –3
    |    wgrib   hoge.grib   -i   -ieee   -nh   -o   out.bin
```



Oh, and can I do this too?

```
set ofile = slp-jan1000.grib
rm $ofile
foreach   yr    ( 1999, 2000, 2001, 2002)
  wgrib slp${yr}.grib  | grep “1000hPa”  | grep “:U=1:” > tmp.list
```

GRIB は cat で合成できる

```
wgrib z1000.grib
1:0:d=01010100:HGT:kpds5=7:kpds6=100:kpds7=1000:TR=113:P1=0:P2=6:TimeU=1:1000 mb:anl:ave@6hr:NAve=124
2:15852:d=01020100:HGT:kpds5=7:kpds6=100:kpds7=1000:TR=113:P1=0:P2=6:TimeU=1:1000 mb:anl:ave@6hr:NAve=112
wgrib z850.grib
1:0:d=01010100:HGT:kpds5=7:kpds6=100:kpds7=850:TR=113:P1=0:P2=6:TimeU=1:850 mb:anl:ave@6hr:NAve=124
2:17166:d=01020100:HGT:kpds5=7:kpds6=100:kpds7=850:TR=113:P1=0:P2=6:TimeU=1:850 mb:anl:ave@6hr:NAve=112

cat  wgrib z1000.grib z850.grib  > new.grib
wgrib new.grib
1:0:d=01010100:HGT:kpds5=7:kpds6=100:kpds7=1000:TR=113:P1=0:P2=6:TimeU=1:1000 mb:anl:ave@6hr:NAve=124
2:15852:d=01020100:HGT:kpds5=7:kpds6=100:kpds7=1000:TR=113:P1=0:P2=6:TimeU=1:1000 mb:anl:ave@6hr:NAve=112
3:31704:d=01010100:HGT:kpds5=7:kpds6=100:kpds7=850:TR=113:P1=0:P2=6:TimeU=1:850 mb:anl:ave@6hr:NAve=124
4:48870:d=01020100:HGT:kpds5=7:kpds6=100:kpds7=850:TR=113:P1=0:P2=6:TimeU=1:850 mb:anl:ave@6hr:NAve=112
```

GRIB を GrADS で直接見る

必要なもの

GRIB ファイル本体

ctl ファイル → grib2ctl.pl で自動作成

idx ファイル → gribmap コマンドで自動作成

```
grib2ctl.pl -ncep_rean -no_prs hoge.grib > hoge.ctl
gribmap hogr.ctl -t0
```

ctl ファイル内の時間とのつじつまをあわせる。 -0 オプションでもよい

Part3: NetCDF

NetCDF は自己記述データ

データ自身の中に、データの情報が書き込まれている

つまり、「データと ctl ファイル」のように、データの中身についての対応表を別に持つ必要がない

NetCDF は何にでも応用できる

主として「緯度、軽度、高度、時間」の次元のデータに利用されるが、この次元を「ステーション番号」にすることで地点データも扱える

NetCDF はちょっと面倒

必要な知識が多い。
逆に知っていると幸せな利用方法がいくらでもある。

NetCDF を直接 GrADS で見る

条件： NetCDF ファイルがCOARDS の規格に準拠して作られていること

```
grads> sdfopen slp.mon.mean.nc
```

NetCDF のインストール

- Unidata (http://www.unidata.ucar.edu) からソースをダウンロード
- コンパイルする

```
tar xZvf netcdf-3.5.tar.Z
cd netcdf-3.5/src
setenv  CPPFLAGS  -Df2cFortran
./configure -prefix=/usr/local
gmake
gmake install    ←昔はここでコケた。今は大丈夫
```

- 必要なら、udunits もダウンロードしてインストールする。これは NetCDF の作成に必要なライブラリ

ncdump で構造をみる

次元はここ

変数はここ

offset と
scale_factor

```
ncdump -h slp.mon.mean.nc

netcdf slp.mon.mean {
dimensions:
        lon = 144 ;
        lat = 73 ;
        time = UNLIMITED ; // (650 currently)
variables:
        float lat(lat) ;
                lat:units = "degrees_north" ;
                lat:actual_range = 90.f, -90.f ;
                lat:long_name = "Latitude" ;
        float lon(lon) ;
                lon:units = "degrees_east" ;
                lon:long_name = "Longitude" ;
                lon:actual_range = 0.f, 357.5f ;
        double time(time) ;
                time:units = "hours since 1-1-1 00:00:0.0" ;
                time:long_name = "Time" ;
                time:actual_range = 17067072., 17541192. ;
                time:delta_t = "0000-01-00 00:00:00" ;
                time:prev_avg_period = "0000-00-00 06:00:00" ;
        float slp(time, lat, lon) ;
                slp:long_name = "Sea Level Pressure" ;
                slp:valid_range = 870.f, 1150.f ;
                slp:actual_range = 962.8222f, 1082.558f ;
                slp:units = "millibars" ;
                slp:add_offset = 0.f ;
                slp:scale_factor = 1.f ;
                slp:missing_value = -9.96921e+36f ;
                slp:precision = 1s ;
                slp:least_significant_digit = 1s ;
                slp:var_desc = "Sea Level Pressure¥n",
```

NetCDF 読み取りプログラムの要点

1. 配列宣言

```
parameter(ilon=144, ilat=73)
 real*4     rdata(ilon,ilat,1,1)
 integer*2 sdata(ilon,ilat,1,1)
```

読み込むデータが float なら real*4 を、short なら、
integer*2 の配列を用意する
データが4次元なら、4次元配列を用意する

2. netcdf.inc を読み込む

```
 include '/usr/local/include/netcdf.inc'
```

これはインストールのしかたによって場所が違う。
インストール時に –prefix=/usr/local/ を指定したなら
これでOK

NetCDF 読み取りプログラムの要点(2)

3. 準備をする

```
sta = nf_open('slp.nc',0,fileid)
sta = nf_inq_varid(fileid,'slp',varid)
sta = nf_get_att_real(fileid,varid,'scale_factor',scale)
sta = nf_get_att_real(fileid,varid,'add_offset',offset)
```

ファイル名と、その内部の変数名を使って ID を取得。
さらに scale と offset も取得しておく

4. 読み込み領域を指定する

```
data start /1,1,6,1/          これは始点
data range /144,73,1,1/       終わりの点に向かってのベクトル
```

始点と終点ではないのに注意。
この行列を変えれば、経度時間断面なども一発でとれる

NetCDF 読み取りプログラムの要点(3)

5. 読み込み

```
sta = nf_get_vara_real(fileid,varid,start,range,rdata)
sta = nf_get_vara_int2(fileid,varid,start,range,sdata)
```

選んだ断面を配列に、一気に読み込む

6. スケールとオフセットを処理する

```
do i = 1,ilat
  do j = 1,ilon
    data(j,i) = 1.0 * sdata(j,i,1,1) * scale + offset
  end do
end do
```

たとえば気温のデータがケルビンで入っていたら、
オフセットは -273 になっているはず。

NetCDF 読み取りプログラムの要点(4)

7. コンパイル

```
% f77 -w read.f  -o read.out  -lnetcdf  -L/usr/local/lib
% ./read.out
```

libnetcdf.a というライブラリをリンクしないといけないので、-lnetcdf というオプションをつける。
ライブラリの置き場所を f77 が知らない場合は、-L オプションも必要かもしれない。

NCO： NetCDF Operators

NetCDF を扱うためのコマンドラインツール
wgrib に近いことをこれでできる

- ncatted – attribute editor
- ncdiff – differencer
- ncea – ensemble averager
- ncecat – ensemble concatenator
- ncflint – file interpolator
- ncks – kitchen sink (extract, cut, paste, print data)
- ncra – running averager
- ncrcat – record concatenator
- ncrename – renamer
- ncwa – weighted averager

自分のPCで解析をする

- OS は Linux, FreeBSD, IntelSolaris, Windows
- Fortran コンパイラ → f2c + 沼口さんパッチで十分
- NetCDF は上記のどれでもインストール可
- GrADS もどれでも動く。(FreeBSD は Linux エミュレーションで)
- GMT もすぐにコンパイルできる
- 画像処理用に、ghostscript, gv, xv, ImageMagick をいれる
- 論文作成用に pLaTeX2e をいれる （FreeBSD, Linux では自動）

メリット:

- マシンを独り占め
- 今の6万円のPCなら、分野のどのWSよりも速い
- 研究環境のすべてを自分の管理下における
- 技術を学べば、作業は速くなるし

Sep. 13, 2002 M.E.H.