GrADSでのバイナリデータの取り扱い

予備知識

ビット

1ビット（bit）はコンピュータが扱う最小単位で、2進数の1桁を表す

1ビットのデータで，0か1の2つの数値を表すことができる。

バイト

ビットを8個集めたデータを1バイト（Byte）と呼ぶ。

1バイトで2進数の8桁を扱うことができる。

2進法で表現すると, 1バイト = 00000000 ～ 11111111までの数値

10進法で表現すると，1バイト=0～256 (=2^8-1) までの数値

現在のコンピューターは1バイトをひと固まりのデータとして，処理を行っている

2進法と16進法

16進法を用いると, 2進数で4桁の数字を1桁で表すことができるので, 人間が見る場合には, 桁数の多くなる2進法ではなく16進法を用いることが多い。

10進数	0	1	2	3	4	5	6	7	8	9
2進数	0	1	10	11	100	101	110	111	1000	1001
16進法	0	1	2	3	4	5	6	7	8	9

10進数	10	11	12	13	14	15
2進数	1010	1011	1100	1101	1110	1111
16進法	A	B	C	D	E	F

16進法での1バイトのデータの表記法

1バイトは, 0～2^8-1の数値を表すことができる (0000 0000～1111 1111)

前後半の4桁は, それぞれ16進法で表すことができる

10進数	10	11	12	13	14	15
2進数	1010	1011	1100	1101	1110	1111
16進法	A	B	C	D	E	F

したがって，2進法の8桁の数値は16進法を使って下表のように表すことができる

2進数	0000 1010	0000 1011	0000 1100	0000 1101
16進法	0A	0B	0C	0D

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

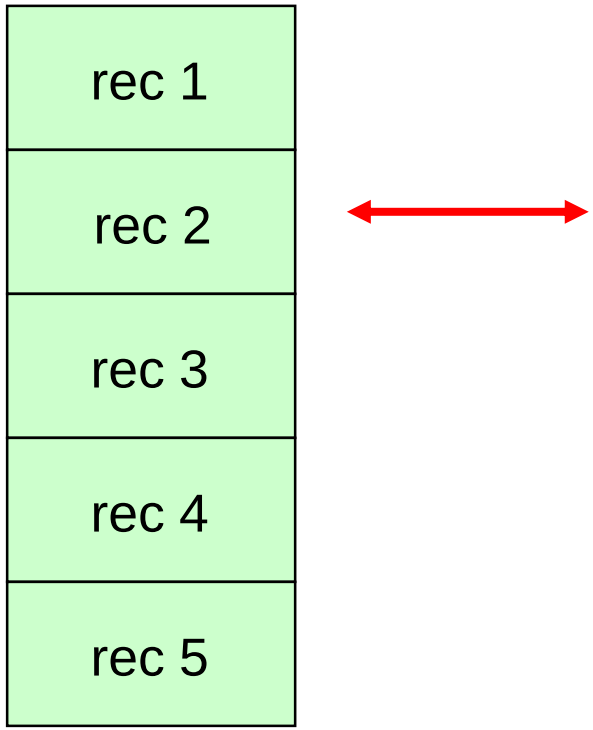
データは小さくなる

GrADSが採用しているデータ構造

概要

GrADSはCTL (コントロール)ファイルを使って，データ構造を把握する

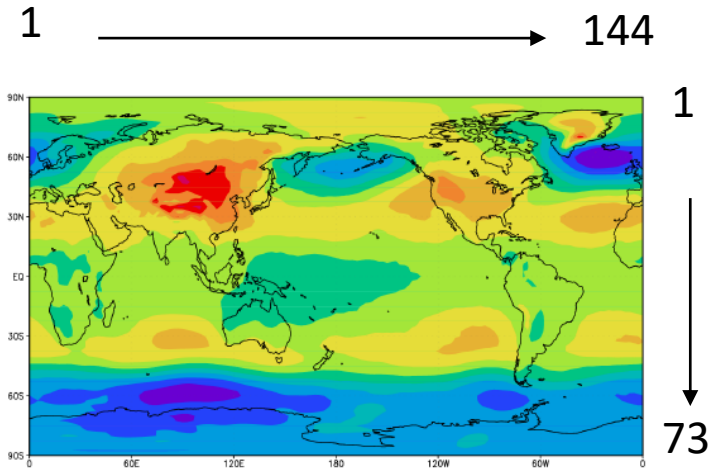
データが記憶されている
バイナリファイル



対応するCTLファイル

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

描画イメージ



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

※CTLファイルはデータ作成者（利用者）が自分で作成する

GrADS データの構造(1)

1変数 (Z)、1層、時間方向にだけ複数のレコードがある(5個)

データが記憶されている
バイナリファイル

t=1におけるz (144×73要素)
t=2におけるz (144×73要素)
t=3におけるz (144×73要素)
t=4におけるz (144×73要素)
t=5におけるz (144×73要素)

レコード1
レコード2
レコード3
レコード4
レコード5

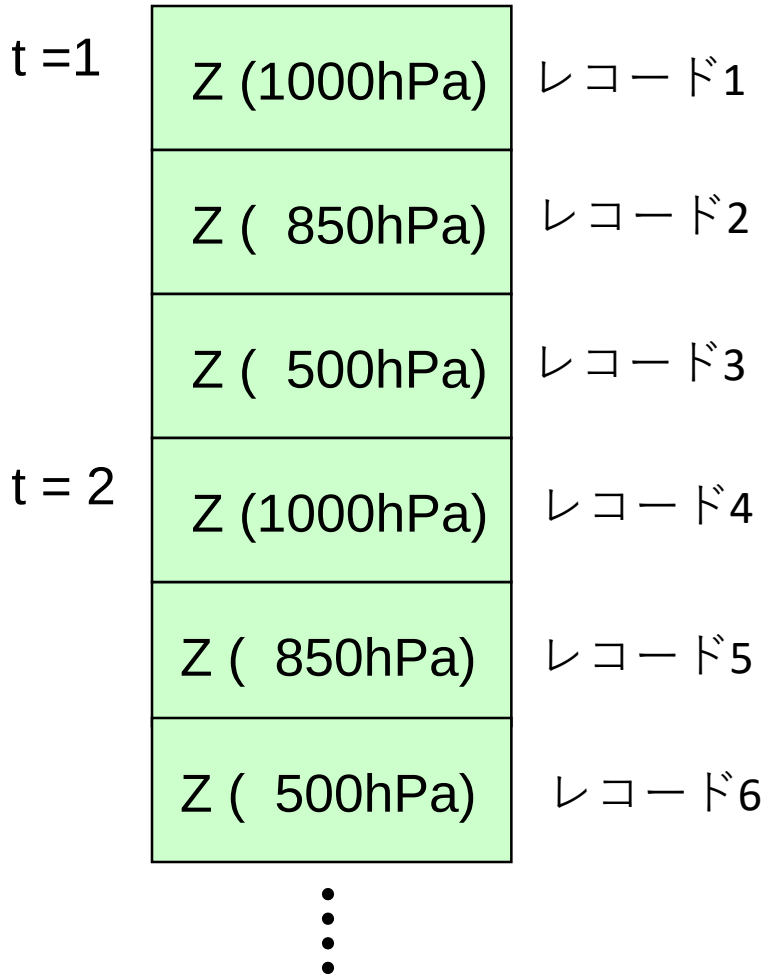
vars 1: 変数の数=1 (zのみ)
tdef 5: 時間方向に5個のデータ

対応するCTLファイル

```
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



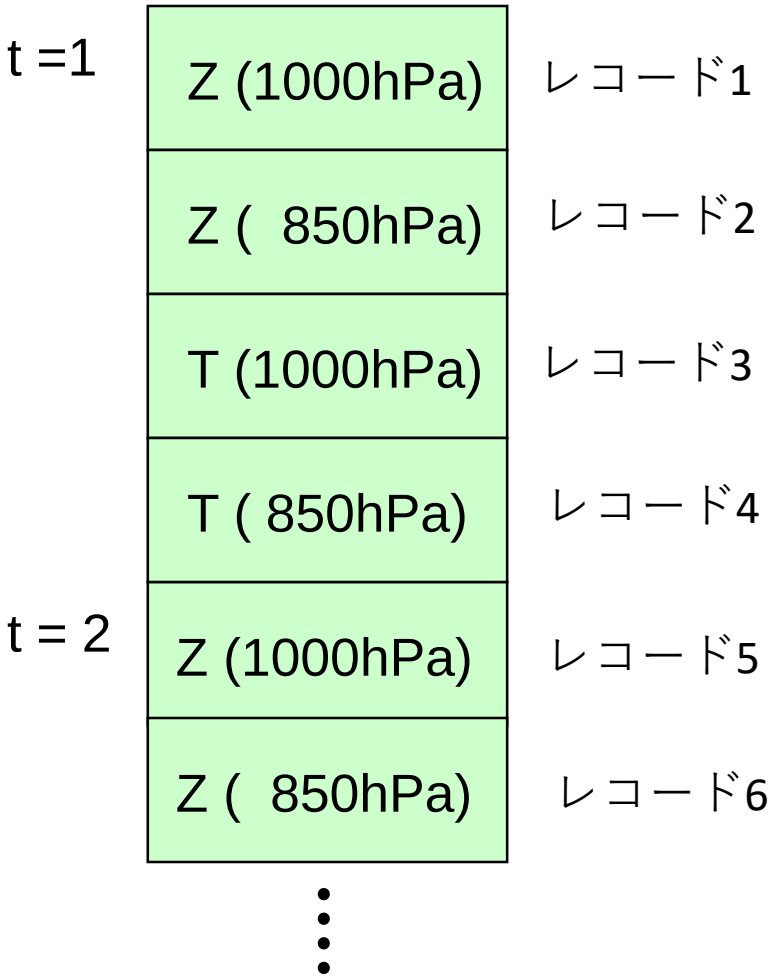

```
dset ^sst.mon.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
tdef   1 linear 00Z01nov1981 1mo
zdef   3 levels 1000 850 500
vars   1
slp    3 0 Sea Level Pressure
endvars
```

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
```

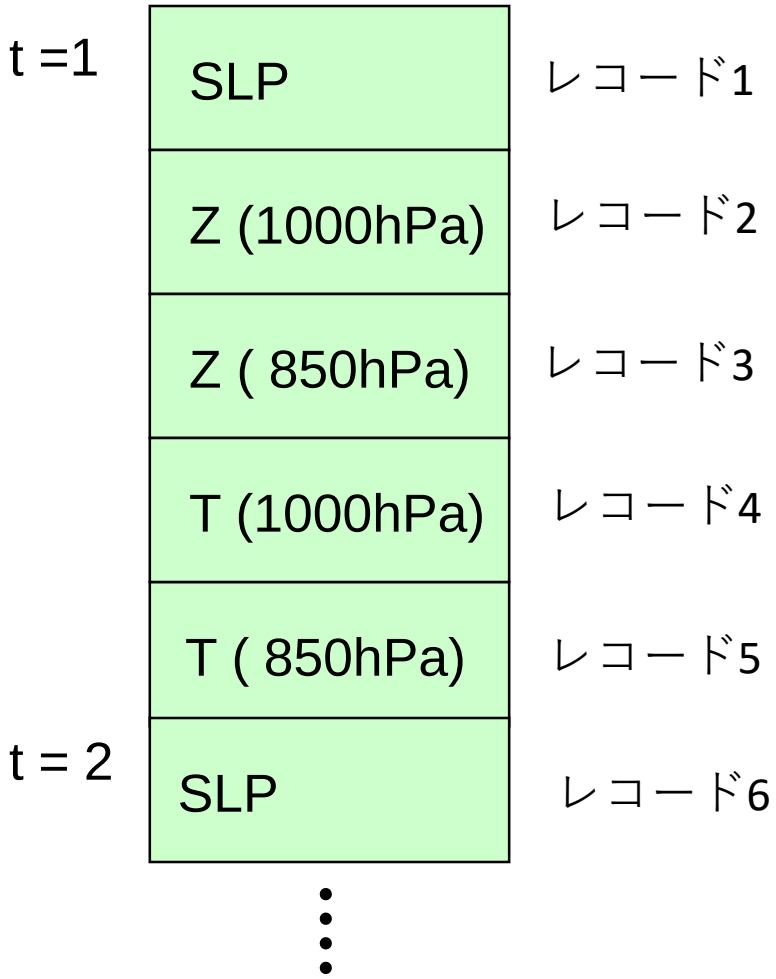
隣り合ったレコードは層が違う。

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



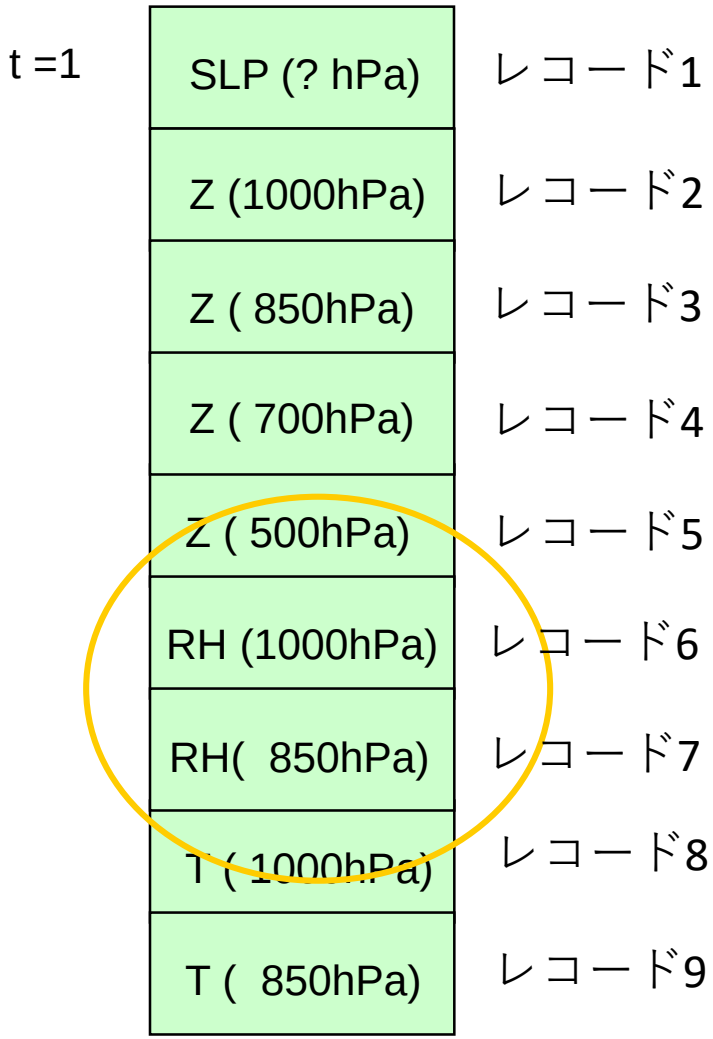

```
dset ^sst.mon.bin
undef -999
options yrev big_endian
xdef 144 linear   0 2.5
ydef  73 linear -90 2.5
tdef   2 linear 00Z01jan1981 1mo
zdef   2 levels 1000 850
vars   3
slp  0 0 Sea Level Pressure
z    2 0 Geopotential Height
t    2 0 Temperature
endvars
```

表層データは　0 層のデータ

※SLP=海面気圧

GrADS データの構造(5)

一部の要素は層が少ない場合




```
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 データの読み込み

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

ファイルを開く

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

読みこみ

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

レコード1に記録されているすべてのデータが配列data(ilon,ilat)に読み込まれる

GrADS データの書きこみ

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

ファイルを開く

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

書きこみ

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

レコード1に配列data(ilon,ilat)に含まれるすべてのデータが書き込まれる

エンディアン

16進法での1バイトのデータの表記法

1バイトは, 0〜2^8 の数値を表すことができる (0000 0000〜1111 1111)

前後半の4桁は, それぞれ16進法で表すことができる

10進数	10	11	12	13	14	15
2進数	1010	1011	1100	1101	1110	1111
16進法	A	B	C	D	E	F

したがって，2進法の8桁の数値は16進法を使って下表のように表すことができる

2進数	0000 1010	0000 1011	0000 1100	0000 1101
16進法	0A	0B	0C	0D

BigエンディアンとLittleエンディアン

１バイトのデータは256個の数値が収納できる。

1バイトのデータは, 16進法で00 ～ FF （10進法で0～255 ）のという数字まで表現できる。それ以上の数値は，1バイトをひと固まりとして，メモリに収納していく

ところが，各桁の数値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

例として0A 0B 0C 0Dという数値を考える。

Big Endian

0A 0B 0C 0D と、**大きい桁を先に**格納する

スーパーコンピューターはこちらを用いることが多い

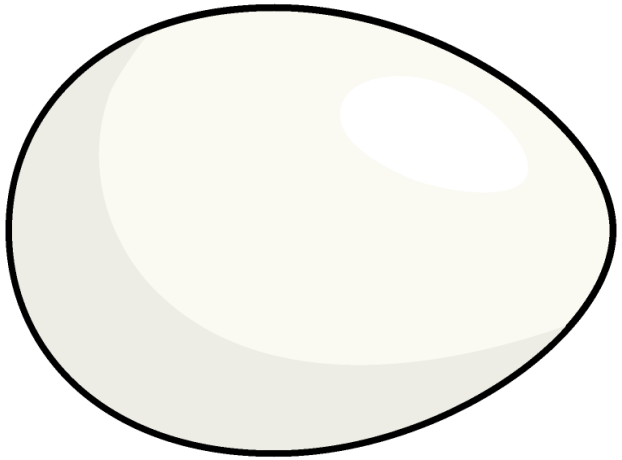
Little Endian

0D 0C 0B 0A と、**小さい桁を先に**格納する

市販のパソコンはこちらを用いるものが多い

語源

Big Endian



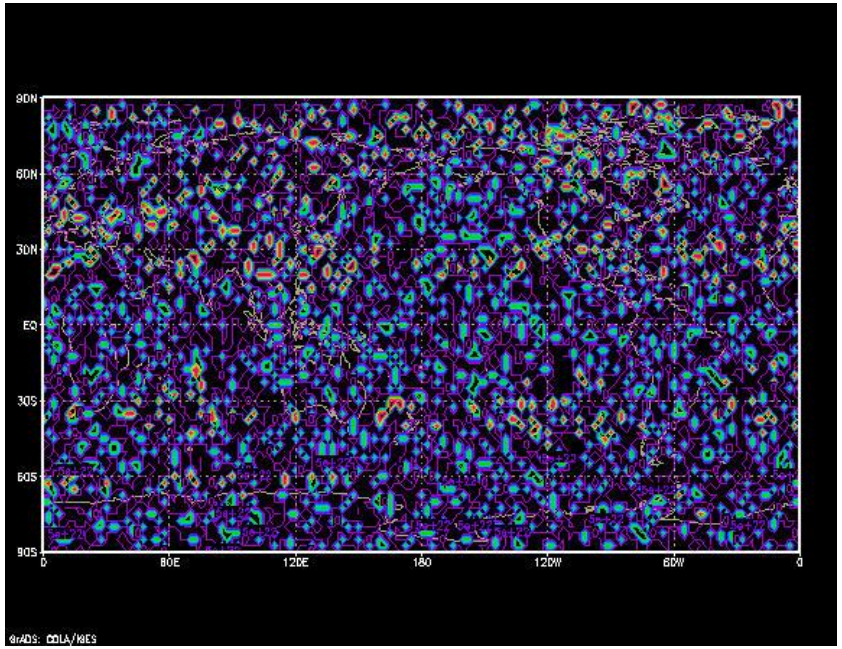
Little Endian

Big Endian のデータをリトルエンディアンのパソコン で読み込むと、

特徴的な、非常に大きい（小さい）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

図を書くと、ぐちゃぐちゃの図になる



CTLファイルでのエンディアンの指定法

ビッグエンディアンのデータを読むとき

options big_endian

リトルエンディアンのデータを読むとき

options little_endian

Fortranプログラムをコンパイルする際のエンディアン指定法

Intel Fortran

Ifort **-convert big_endian**

gfortran

gfortran **-fconvert=big-endian**

付録

GrADSでのプレーンバイナリファイルの作り方：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 gradsのfwrite を使うとよい

注意: 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 (スーパーコンピューターでない限りlittleエンディアンになることがほとんど)

```
open slp.mon.mean.ctl
set x  1 144
set fwrite sample.bin
set gxout fwrite
d slp
```

注意事項

fwrite時のset x

fwriteの際にはxとyのsetが必須

q dimsで表示されるxの格子点数は1つ大きい場合があるので，その場合一つ引いておく（q dims; say resultの結果とCTLファイルと比較して確認しておく）

open ～.ctl

set x 1 144

set y 1 73

set t 1 20

set gxout fwrite

set fwrite -le hoge.bin

d var