0.5 ジュニア版　　プログラミングの技術

プログラムを組み立てるには、「ループで回す」ことと「ＩＦで分岐させる」ことの２つの技術が必要です。今までのプログラムは、単なる定義の定義を繰り返していただけです。ループとＩＦ分岐に入る前に、GAUSS のプログラムで重要になってくる基礎概念を説明しておきます。まずは、行列配列のインデックス番号と単位行列および零行列をみます。

プログラム

```
new;
cls;
X={9 7 2,
   7 8 2,
   2 2 8};
I=eye(3);
X_1=inv(X);
O=zeros(3,3);
print "X=" X;
print "x32=" x[3,2];
print "Row 1=" x[1,.];
print "Column 2=" x[.,2];
print "I=" I;
print "O=" O;
print "X_1=" X_1;
print/rd "X_1*X=" X_1*X;
print/rd "X*X_1=" X*X_1;
print "I*X=" I*X;
print "X*I=" X*I;
print "O*X=" O*X;
print "X'=" X';
print "|X|=" det(X);
print "Chol(X)=" chol(X);
print "Chol(X)'Chol(X)=" chol(X)'chol(X);
```

画面表示

```
X=                          ２列目
        9.0000000       7.0000000       2.0000000
        7.0000000       8.0000000       2.0000000
        2.0000000       2.0000000       8.0000000   ３行目
```

x32= 2.0000000

Row 1= 9.0000000 7.0000000 2.0000000

Column 2=

7.0000000

8.0000000

2.0000000

I=

1.0000000	0.00000000	0.00000000
0.00000000	1.0000000	0.00000000
0.00000000	0.00000000	1.0000000

O=

0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000

X_1=

0.34883721	-0.30232558	-0.011627907
-0.30232558	0.39534884	-0.023255814
-0.011627907	-0.023255814	0.13372093

X_1*X=

1.00000000	0.00000000	0.00000000	
0.00000000	1.00000000	0.00000000	= I
0.00000000	0.00000000	1.00000000	

X*X_1=

1.00000000	0.00000000	0.00000000	
0.00000000	1.00000000	0.00000000	= I
0.00000000	0.00000000	1.00000000	

I*X=

9.0000000	7.0000000	2.0000000
7.0000000	8.0000000	2.0000000
2.0000000	2.0000000	8.0000000

X*I=

9.0000000	7.0000000	2.0000000
7.0000000	8.0000000	2.0000000
2.0000000	2.0000000	8.0000000

O*X=

0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000
0.00000000	0.00000000	0.00000000

X'=

9.0000000	7.0000000	2.0000000	
7.0000000	8.0000000	2.0000000	＝X（この場合対称行列）
2.0000000	2.0000000	8.0000000	

|X|= 172.00000

Chol(X)=

3.0000000	2.3333333	0.66666667	
0.00000000	1.5986105	0.27801922	行列の「ルート」
0.00000000	0.00000000	2.7346409	

Chol(X)'Chol(X)=

9.0000000	7.0000000	2.0000000	
7.0000000	8.0000000	2.0000000	＝X
2.0000000	2.0000000	8.0000000	

プログラム各行の説明

１行目　メモリを初期化する
２行目　スクリーンをクリアする
３～５行目　行列の要素を設定して、変数Ｘに入れる（カンマまでが１行）
６行目　３×３のディメンションの単位行列Ｉを組込み関数 eye で作成し変数Ｉに入れる
７行目　組込み関数 inv でＸの逆行列を作成し、変数Ｘ_1 に入れる
８行目　組込み関数 zeros を用いて３×３の単位行列を作成し、変数Ｏに入れる
９行目　変数Ｘの内容を、引用符内のメッセージとともに、画面表示させる
10 行目　X[3,2]つまり行列Ｘの３行２列目の要素を見つけた上で、画面表示させる（なお GAUSS では[　]には配列、{　}には要素、(　)には関数のインプットを入れる）
11 行目　X[1,.]つまり行列Ｘの１行目の行ベクトルを見つけた上で、画面表示させる
12 行目　X[.,2]つまり行列Ｘの２列目の列ベクトルを見つけた上で、画面表示させる
13 行目　上で計算した変数Ｉの内容（単位行列）を説明メッセージ付で画面表示する
14 行目　上で計算した変数Ｏの内容（零行列）を説明メッセージ付で画面表示する
15 行目　上で計算した変数Ｘ_1 の内容（Ｘの逆行列）を画面表示する
16 行目　上で定義した２つの変数を用いて、$X^{-1}X$を計算してその結果を画面表示する
17 行目　同様にして、XX^{-1}を計算してその結果を画面表示する
18 行目　同様にして、ＩＸを計算してその結果を画面表示する
19 行目　同様にして、ＸＩを計算してその結果を画面表示する

20 行目 同様にして、ＯＸを計算してその結果を画面表示する
21 行目 Ｘの転置行列を「'」をつけて求めその結果を画面表示する（Ｘそのものになっているので、このＸは対称行列）
22 行目 組込み関数 det を用いてＸの行列式を求めその結果を画面表示する
23 行目 組込み関数 chol を用いてＸの Cholesky 分解を計算してその結果を画面表示する（なお、この分解計算は行列が対象行列で Positive Definite の場合のみ計算可）
24 行目 Chol(Ｘ)'Chol(Ｘ)を計算してその結果を画面表示する（Ｘそのものになる）なお、転置の印がついているときは行列の掛け算の記号＊は省略できる

上では、GAUSS のプログラムの基礎となる行列の扱いを示すいろいろな事項を一度に計算し画面表示させています。変数にスケラー（１×１の行列）を入れるのであれば、通常どおりイコールサインで変数と関数を X=1;というようにつないで「後ろから前に数値を入れる」作業をしてやれば設定できます。これがベクトルや行列になると、{　}括弧を用いてその中に「行ごとにカンマで区切って」入れてやります。ただし、{1,2,3,4,5}は数学では一続きの集合の要素を表して行ベクトルと定義してありますが、縦に並ぶデータを扱う GAUSS では{1,2,3,4,5}は５行１列の列ベクトルを表します。また、GAUSS では変数Ｘと x の大文字の区別は全くありません。変数部分を除いて、通常は小文字で書いていくのが慣習になっています。GAUSS で配列を表すには[　]括弧の中にＸ[1,1]というように書きます。この場合は１行１列目のＸの要素という意味です。また、GAUSS ではＸ[1,.]と書いて１行目のすべて（列はワイルドカードですべての列の要素）という書き方をして行列から行ベクトルを取り出すことができます。同様に、Ｘ[.,1]と書いて１列目のすべてを意味し、この場合には行列Ｘの１列目の要素からなる列ベクトルを取り出せます。行列の基本関係として

$X^{-1}X = XX^{-1} = I$
$IX = XI = X$
$OX = O$
$X' = X$ならば「Ｘは対称行列」
Chol(Ｘ)'Chol(Ｘ)＝Ｘ（ただし、Ｘは対称行列で Positive Definite）

ということが言えます。それを確かめているのが上のプログラムです。逆行列（インバース）とは、かけて単位行列Ｉになるもの。その単位行列Ｉとは、それをある行列にかけてそのもの自身になるもので、対角要素（ダイアゴナル）がすべて１で、その他の要素（オフアゴナル）がすべて０となるような行列です。通常の数値が０に数値をかけて０になると同じように、すべての要素がゼロからなる零行列に行列をかけても零行列になります。行と列を入れ替えた転置行列ともとの行列が等しいのであれば、それは対称行列を意味します。行列が通常の数値の正の値に相当する「対称かつ Positive Definite」であるならばそ

の行列に対する「行列のルート」とでも呼べる Cholesky 分解ができます。こうして分解したもの同士を（前のものを転置させた上で）かけ合わせると、その意味からもわかるように、その行列そのものになります。なお、Positive Definite の概念は難しいのですが、簡単に言うと、その正方行列の行列式およびそれ以下の次数の小行列の行列式（データミナント）がすべて正になるものを言います。この場合、もとの行列は３×３ですから、この行列式を見て正であることが計算されています。さらに、２×２の行列を作って行列式が正であるかをすべて確認します。こういう作業を繰り返してすべて正になるものを Positive Definite とよばれる「行列の正の値」とも呼べる概念です。

今度は、プログラムの変数設定によく使われる「１からできた行列」の使い方とこれまでにも少し取り上げた水平方向および垂直方向のマージについてみます。また、GAUSS 独自の「列±スケラー」の計算を説明します。

プログラム

```
new;
cls;
c=5*ones(5,1)|105;
print ones(3,3);
print "c={5,5,..,5,105}=" c;
print (1~2)|(3~4);
print "seqa(1,1,5)=" seqa(1,1,5);
print "c -3=" c-3;
print "c'-3=" c'-3;
```

画面表示

```
      1.0000000      1.0000000      1.0000000
      1.0000000      1.0000000      1.0000000
      1.0000000      1.0000000      1.0000000
c={5,5,..,5,105}=
      5.0000000
      5.0000000
      5.0000000
      5.0000000
      5.0000000
      105.00000

      1.0000000      2.0000000
      3.0000000      4.0000000
```

```
seqa(1,1,5)=
        1.0000000
        2.0000000
        3.0000000
        4.0000000
        5.0000000
c -3=
        2.0000000
        2.0000000
        2.0000000
        2.0000000
        2.0000000
        102.00000
c'-3=   2.0000000   2.0000000   2.0000000   2.0000000   2.0000000   102.00000
```

プログラム各行の説明

１行目　メモリを初期化する

２行目　スクリーンをクリアする

３行目　組込み関数 ones を使って５×１の「１ばかりの」列ベクトルを作成しておいて、これに５をかけたものを（すなわち５ばかりの列ベクトル）を 105 と垂直方向にマージしたものを変数ｃに入れる

４行目　３行３列の１ばかりの要素の行列を作成して、これを画面表示する

５行目　上で設定した変数ｃの内容を、引用符内のメッセージとともに、画面表示する

６行目　１と２を水平方向に「～」を使ってマージしたものと、３と４を同様にして水平方向にマージしたもののの２つを、垂直方向に「｜」を使ってマージする

７行目　組込み関数 seqa を用いて１から１ステップで５個分のシークエンスを作成してこれを説明メッセージとともに画面表示する

８行目　上で設定した列ベクトルｃからスケラー３を引くと、GAUSS では自動的に３が引かれる方のベクトルのディメンションと同じになって、すべての要素から３が引かれる

９行目　同様にして、ｃを転置した行ベクトルであっても、引かれる側のディメンションに自動的に３が伸びて、すべての要素から３が引かれる

GAUSS で初期値などの列ベクトルを設定する場合には、しばしば垂直方向のマージ「｜」の記号を使います。また、そのベクトルの要素に同じ数がたくさん続く場合には１ばかりからなるベクトルにその数をかけることによって繰り返しを省略します。つまり、例えばもっと極端な例で、{8,8,…..,8,108}というふうに８が 29 回続き、最後が 108 のケースは、

```
8*ones(29,1)|108
```

というふうに GAUSS では書くことができます。29×１の１ばかりの要素からなる列ベクトルに８をかけることで８ばかりの列ベクトルにしておき、そこに垂直方向に 108 をマージして、最終的に 30×１の列ベクトルを作成します。こうしたシークエンスベクトルの作り方は、通常の増加していくシークエンスを作成する組込み関数 seqa とともに、同じパターンのベクトル入力を簡素化するものです。また、こうした方法を採用する方が、手で入力していくよりも間違いが少なくなります。最後のベクトルからスケラーを引くところですが、これは下のように３というスケラー（１×１の行列）が前の引かれる側のディメンションにあわせて６×１のベクトルになって、

$$\begin{bmatrix}5\\5\\5\\5\\5\\105\end{bmatrix}-3\Rightarrow GAUSS\Rightarrow\begin{bmatrix}5\\5\\5\\5\\5\\105\end{bmatrix}-\begin{bmatrix}3\\3\\3\\3\\3\\3\end{bmatrix}$$

と変換された上で計算がなされるということを意味しています。これは、もうすでに分散を求める関数の自作のところで少しふれたことです。データ列からその列の平均の値を引く時に、わざわざ引く側の平均値をベクトルに直さなくてもよいということです。同様に

$$\begin{bmatrix}5&5&5&5&5&105\end{bmatrix}-3\Rightarrow GAUSS\Rightarrow\begin{bmatrix}5&5&5&5&5&105\end{bmatrix}-\begin{bmatrix}3&3&3&3&3&3\end{bmatrix}$$

というふうに、行ベクトルとスケラーの場合も同じように自動的に変換がなされます。このことは引き算だけでなく足し算にも適用されます。

上の最後で行なった「ベクトル±スケラー」の話をさらにすすめて、「行ベクトル±列ベクトル」（またはその逆）について具体例をプログラムします。

プログラム

```
new;
cls;
x=seqa(1,1,5);
y=zeros(1,2);
print x+y;
a=seqa(1,1,5)';
b=zeros(5,1);
print a+b;
```

```
x1={1,2,3};
x2={1 2 3};
print x1+x2;
print x1^2+x2^2;
```

画面表示

```
1.0000000    1.0000000
2.0000000    2.0000000
3.0000000    3.0000000
4.0000000    4.0000000
5.0000000    5.0000000

1.0000000    2.0000000    3.0000000    4.0000000    5.0000000
1.0000000    2.0000000    3.0000000    4.0000000    5.0000000
1.0000000    2.0000000    3.0000000    4.0000000    5.0000000
1.0000000    2.0000000    3.0000000    4.0000000    5.0000000
1.0000000    2.0000000    3.0000000    4.0000000    5.0000000

2.0000000    3.0000000    4.0000000
3.0000000    4.0000000    5.0000000
4.0000000    5.0000000    6.0000000

2.0000000    5.0000000    10.000000
5.0000000    8.0000000    13.000000
10.000000    13.000000    18.000000
```

上のプログラムでは以下の計算をやらせて、自動的に列と行のの足し算（あるいは引き算）が自動的に行列に展開する様子を示したものです。すなわち、

$$\begin{bmatrix}1\\2\\3\\4\\5\end{bmatrix}+\begin{bmatrix}0 & 0\end{bmatrix}\Rightarrow GAUSS\Rightarrow\begin{bmatrix}1 & 1\\2 & 2\\3 & 3\\4 & 4\\5 & 5\end{bmatrix}+\begin{bmatrix}0 & 0\\0 & 0\\0 & 0\\0 & 0\\0 & 0\end{bmatrix}$$

というふうに、足し算および引き算の場合、足される側と足す側の行数と列数が自動的に同じになって、上のケースではどちらも５×２の行列になってから両者が足し合わされて

います。同じような例として、次のケースは、

$$\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix}+\begin{bmatrix}0\\0\\0\\0\\0\end{bmatrix}\Rightarrow GAUSS\Rightarrow\begin{bmatrix}1 & 2 & 3 & 4 & 5\\1 & 2 & 3 & 4 & 5\\1 & 2 & 3 & 4 & 5\\1 & 2 & 3 & 4 & 5\\1 & 2 & 3 & 4 & 5\end{bmatrix}+\begin{bmatrix}0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\\0 & 0 & 0 & 0 & 0\end{bmatrix}$$

というふうに今度は行ベクトルに列ベクトルを足し合わせたものです。その前の列ベクトルに行ベクトルを足し合わせたのと全く同じ要領です。これらの計算は、GAUSS 独自の計算であって、数学的な行列やベクトルの計算とは無関係です。しかしながら、これらのコンピュータ計算上の約束事は、上のプログラムの一番最後に示した次のような例で特に重要になってきます。

$$\begin{bmatrix}1\\2\\3\end{bmatrix}+\begin{bmatrix}1 & 2 & 3\end{bmatrix}\Rightarrow GAUSS\Rightarrow\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix}+\begin{bmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{bmatrix}$$

$$\begin{bmatrix}1\\2\\3\end{bmatrix}\text{\^{}}2+\begin{bmatrix}1 & 2 & 3\end{bmatrix}\text{\^{}}2\Rightarrow GAUSS\Rightarrow\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix}\text{\^{}}2+\begin{bmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{bmatrix}\text{\^{}}2$$

上の列ベクトルと行ベクトルの足し算は、これまでと全く同じ要領で、GAUSS 上では、両者の行数と列数に同じディメンションの行列に同じ行または列からなるものを用いて伸ばした上で足し合わせます。零ベクトルでなくても同じことです。問題は、次のべき乗やこの場合ついていませんがさらに何かがかけ合わせている場合の列ベクトルおよび行ベクトルを足し合わせる場合です。上の場合、１から始まる座標計算の役割を果たしています。すなわち、上のべき乗計算の足し算で得られた結果は、Ｘ座標およびＹ座標が 1,2,3 とそれぞれ与えられた時の高さ（Ｚ座標）の値を計算していることになるのです。少し考え方が厄介ではありますが、列ベクトルと行ベクトルの足し算を応用した、この座標軸の作り方は３次元のグラフを書いたり３次元の表面を計算したりする際に利用されます。

Do while ループについて

いろいろなループの作り方がありますが、ここでは最も基本となる do while 命令に絞って、そのループのさせ方と、それにまつわる 1 ずつ増えていく「カウンタ」という変数の管理の仕方を説明します。

まずは、１を５回足し合わせてた合計を計算する方法です。

プログラム

```
new;
cls;
S=0;
i=1;
do while i<=5;
    S=S+1;
    i=i+1;
endo;
print "1+1+1+1+1=" S;
```

画面表示

```
1+1+1+1+1=          5.0000000
```

プログラムの構造

Ｓ＝０

ｉ＝１（カウンタ初期設定）

ループ１回目（カウンタｉ＝１　≦５）

　Ｓ←０＋１

　カウンタ　ｉ←１＋１

ループ２回目（カウンタｉ＝２　≦５）

　Ｓ←１＋１

　カウンタ　ｉ←２＋１

ループ３回目（カウンタｉ＝３　≦５）

　Ｓ←２＋１

　カウンタ　ｉ←３＋１

ループ４回目（カウンタｉ＝４　≦５）

　Ｓ←３＋１

　カウンタ　ｉ←４＋１

ループ５回目（カウンタｉ＝５　≦５）

　Ｓ←４＋１

　カウンタ　ｉ←５＋１

ループ６回目（カウンタｉ＝６　≦５**ではない**）　⇒　ループから抜ける

次は、そうではなくて１から順に１ずつ増えていく数を順に５回分加えた合計を計算するプログラムです。上の１のところがｉに変わります。

プログラム

```
new;
cls;
S=0;
i=1;
do while i<=5;
    S=S+i;
    i=i+1;
endo;
print "1+2+3+4+5=" S;
```

画面表示

```
1+2+3+4+5=          15.000000
```

プログラムの構造

Ｓ＝０

ｉ＝１（カウンタ初期設定）

ループ１回目（カウンタｉ＝１　≦５）

Ｓ←０＋１

カウンタ　ｉ←１＋１

ループ２回目（カウンタｉ＝２　≦５）

Ｓ←１＋２

カウンタ　ｉ←２＋１

ループ３回目（カウンタｉ＝３　≦５）

Ｓ←３＋３

カウンタ　ｉ←３＋１

ループ４回目（カウンタｉ＝４　≦５）

Ｓ←６＋４

カウンタ　ｉ←４＋１

ループ５回目（カウンタｉ＝５　≦５）

Ｓ←１０＋５

カウンタ　ｉ←５＋１

ループ６回目（カウンタｉ＝６　≦５**ではない**）　⇒　ループから抜ける

以前のケースはＳは常に＋１なのでＳ＝５、この場合は＋ｉなのでＳ＝１５になる。

ループにはこの他、上の Do while ループのちょうど反対の意味に相当する書き方として

```
do until i>5;
```

と書く方法があります。ちょうど i<=5「のうちは」の事象の逆が i>5 ですから「そこまで」の until の場合には符号を逆にして等号を取ったものにします。シークエンスを利用して上の i に相当するカウンタを自動的に管理させるものとして For ループがありますが、混乱をするといけないので、ここでは取り上げないことにします。基本的に、Do while 命令ですべてのプログラムが書けます。また、初心者から中級者の段階で do while によって、自分でカウンタを管理した方がプログラム上達につながります。

注意　近年のプログラム技術の全般に言えるの暗黙の了解は、do while i<=100;というようにカウンタが増加していく場合には、見た目の論理展開と同じように、i と 100 の関係が右側の 100 の方が常に大きくなるように書くのが最も見やすいプログラムとされています。読み手に理解されるプログラムを書くには、do until によって右側が大きくなるいう慣習を逆転させるのではなくて、do while によって常に右側が大きくなるように論理展開させることが重要です（ただし、カウンタが減っていく場合はこの限りにはありません）。自己流プログラムをしている方はお気をつけください。

If 分岐について

ループの他に、プログラムで重要になってくる概念は、If 分岐です。３つのパターンを示しておきましょう。標準正規乱数を１個だけ作り出して、その値に応じて分岐させてみましょう。以前取り上げたように、組込み関数 rndn は標準正規乱数を生成するもので、この場合、１行１列の値を作り出して、まずそれを変数ｘに入れています。

プログラム

```
new;
cls;
x=rndn(1,1);
if x>0;
    print "x is positive." x;
else;
    print "x is not poitive." x;
endif;
```

画面表示

```
x is positive.          0.32621412
```

プログラム

```
new;
cls;
x=rndn(1,1);
if x>0;
    print "x is positive." x;
endif;
```

画面表示

（ｘが０より大きくないときは何もしません）

プログラム

```
new;
cls;
x=rndn(1,1);
if x>0;
    print "x is positive." x;
elseif x<0;
    print "x is negative." x;
else;
    print "x is zero." x;
endif;
```

画面表示

```
x is positive.          0.44119911
```

上の場合は、elseif の部分が１つしかない例ですが、例えばサイコロの目の場合６つありますから、最初は if で対応して、最後を else で対応するものと考えれば、elseif の部分は４つ必要です。すなわち、

```
If 条件文;
    式;
elseif 条件文;
    式;
elseif 条件文;
    式;
elseif 条件文;
    式;
elseif 条件文;
```

```
    式;
else;
    式;
endif;
```

というふうに、６つの式（または命令のブロック）が書けるように、最初の if 部分と最後の else 部分を除いて、合計４つの elseif 部分が必要になってきます。その際、通常の英語の表記のように else if ではなくて、elseif と一語となりますので注意してください。

これらループと If 分岐を組み合わせれば、定義を繰り返すプログラムを超えて、高度なプログラムが可能になります。ループも If 分岐も上の例の場合は、最も簡単な１重の基本形を示しましたが、これが２重にも３重にもなるプログラムとなることが通常です。それらの例については、章が進むにつれて少しずつ扱っていきます。まずは、上であげたことを確実に身につけてください。

注意　プログラムというものは、特に学術プログラムの場合、動けばいいというものではいけません。通常私たちが行なう論理展開と同じように書いていくことが重要になります。そうした場合、「indentation(字下げ)」は重要です。また、print 命令または retp( )命令によって、外部出力または関数の外部に変数が出ていることを明示することも重要です。字下げと外部出力の明示に特に気をつけてプログラムしていってください。おそらくそれが、ある段階で人のプログラムを改良したり丸写ししたりしないで、飛躍的に自分で何でもプログラムできるようになる出発点になることでしょう。

プログラムスタイル

以下の対照表は推奨できるプログラミング法とそうではないプログラミング法をまとめておきます。たは自己流でプログラムを身につけた方々は今も悪い方法でプログラムを公開されておられますから、それらを真似することは極力避けることが大切です。

	正統的な記述法	亜流な記述法
コメント表示	print “This is a comment”;	“This is a comment”;
改行	print;	?; または “ “;
変数値の表示	print a;	a;
カウンタ表現	do while i<=100;	do until i>100;

ループ表現	i=1;
	do while i<=rows(b);
	. . . . . .
	i=i+1;
	endo;
		i=1;do until i>rows(b);
		. . . . . .
		i=i+1;endo;

プログラム記述	new; cls;
	print “a=” kansu(5);
	
	proc kansu(x);
	local n,a;
	. . . . . .
	retp(a);
	endp;
		new; cls;
		“a=” kansu(5);
		proc kansu(x);
		local n,a;
		. . . . . .
		retp(a);
		endp;

まず、print 命令は retp( )命令と並んで外部出力をするという命令であると同時にプログラムの読み手にどこで外部出力が行なわれているかを明示する手段です。print 文はマニュアルにしたがって完全に書きましょう。近年のプログラム技法の標準的な常識では、カウンタが増していく場合、右側が大きくなるように設定します。反対にすると、何重ものル

ープになったときに見とおしが悪くなります。我々が考えている論理展開と全く同じにプログラムもしておきましょう。また、do ループのカウンタは外部で設定します。これもまた、何重ものループになった時に重要になってきます。なお、endo や endif、endp などのブロックの締めの部分は始まりと同じレベルの字下げにして、そこからそこまでがブロックであることを読み手に明示します。最後に、TSP などを経て GAUSS プログラムに移ってこられた方々は、字下げの概念がないかもしれません。この際、きっちりと字下げの概念をマスターしてください。これらの事項は、誰かに指摘されて直した方がいいよと言われるまで直せないかもしれません。また、亜流な方法で直接おそわったし、また有名な論文の付属プログラムもそうなっていると反発するかもしれません。なぜ、重要なのかは、より高度なループが何重にもなるプログラムや複雑なアウトプット構造のプログラムになって、他人のプログラムを丸写ししたり改造移植する作業を超えて、自分で新しいものを考えなければならないときにはじめてその重要性がはっきりとしてきます。

（ただし、GAUSS のソースおよび最近アップロードされた初級および中級のトレーニングマニュアルには、do until の形が多用されています。しかしながら、この方法は我々が通常行なう実際の論理パターンとは違うので初心者にはおすすめできません。）

練習問題

【問１】

Do while を用いて、本文中の１から５までを足し合わせるプログラムを書き直して、１から 100 まで足し合わせるプログラムにしよう。

【問２】

同じく本文中の１から５までを足し合わせるプログラムを書き直して、１から５までをかけ合わせるプログラムにしよう（ヒント：S=1 にしておく必要がある。なぜか？）。

【問３】

問１で取り組んだプログラムを書き直して、１から 100 までの奇数だけを足し合わせるプログラムにさらに書きかえよう（ヒント：１の次は２つ増した３である）。

発展

⇒　本編　3.2　3.3