６　構造化

ここでは、構造化について練習します。構造化とは、頻繁に行う処理をあらかじめ副プログラム（サブルーチンや関数）として主プログラムとは別に作成しておき、主プログラムの中ではサブルーチンや関数を参照するだけで処理を行えるようにすることです。このようにすると、プログラム全体の見通しがよくなり、プログラムの作成を行いやすくなります。はじめに、気温を与えると飽和水蒸気圧を計算して出力するプログラムを、構造化をしないで書いてみます。

プログラム：

FORTRAN

```
C  入力を求めるメッセージを標準出力に書き出す。
      WRITE(6,*) 'Temperature [deg.C]?'

C  READ文で値を読みこむ。
      READ(5,*) T

C  Tを絶対温度に変換する。
      T = T + 273.15

C  飽和水蒸気圧を計算する。
C  EXPは指数関数である。
      ES = 611. * EXP (17.27 * (T-273.16) / (T-35.86))

C  結果を標準出力に書き出す。
C  飽和水蒸気圧の単位はPaなので、hPaに変換するために0.01倍する。
      WRITE(6,*) 'e_s [hPa] =', 0.01*ES

      STOP
      END
```

（参考）C

```
#include <stdio.h>
#include <math.h>

int main( void )
{

/*  変数を宣言する。  */
```

```
  float t, es;

/*  入力を求めるメッセージを標準出力に書き出す。
    "¥n"は改行を表している。  */
  printf( "Temperature [deg.C]?¥n" );

/*  関数 scanf で標準入力から値を読みこむ。
    関数から値を受け取るときは変数名の前に&をつける。  */
  scanf( "%f", &t );

/*  t を絶対温度に変換する。  */
  t = t + 273.15;

/*  飽和水蒸気圧を計算する。
    exp は指数関数である。  */
  es = 611. * exp (17.27 * (t-273.16) / (t-35.86));

/*  結果を標準出力に書き出す。
    飽和水蒸気圧の単位は Pa なので、hPa に変換するために 0.01 倍する。  */
  printf( "e_s [hPa] = %f¥n", 0.01*es );

  return 0;

}
```

実行例：

```
/home/snaoki> f77 prog06_1.f
/home/snaoki> ./a.out
 Temperature [deg.C]?
15.
 e_s [hPa] = 17.0480556
```

飽和水蒸気圧の計算の部分は、関数として別に作成しておくことができます。たとえば、三角関数の値を計算するとき、関数の値を数値的に求めるプログラムを自分で作成しなくても、SIN(THETA)をいうように関数 SIN を使えば sin の値を得ることができました。飽和水蒸気圧の計算のように頻繁に行う計算については、三角関数などと同じように関数という形であらかじめ作成しておくと便利です。しかし、飽和水蒸気圧を計算する関数は FORTRAN や C には組み込まれていないので、自分で定義する必要があります。FORTRAN では主プログラムの後で、副プログラムという形で定義します。今回は実数を返す関数なので、REAL FUNCTION で始まっています。関数名は ES としています。副プログラムは、RETURN 文と END 文で終了します。

プログラム：

FORTRAN

```
C  入力を求めるメッセージを標準出力に書き出す。
      WRITE(6,*) 'Temperature [deg.C]?'

C  READ文で値を読みこむ。
      READ(5,*) T

C  Tを絶対温度に変換する。
      T = T + 273.15

C  結果を標準出力に書き出す。
C  関数ESに引数として温度Tの値を与えると、
C  飽和水蒸気圧の値が返ってくる。
C  飽和水蒸気圧の単位はPaなので、hPaに変換するために0.01倍する。
      WRITE(6,*) 'e_s [hPa] =', 0.01*ES(T)

      STOP
      END

C  主プログラムの後に、関数やサブルーチンのような副プログラムを書く。

C  飽和水蒸気圧を計算するための関数。
C  関数の型は実数(REAL)型、関数名はESとする。
C  引数はTである。
      REAL FUNCTION ES (T)

C  飽和水蒸気圧を計算する。
C  EXPは指数関数である。
        ES = 611. * EXP (17.27 * (T-273.16) / (T-35.86))

C  関数はRETURN文とEND文で終了する。
C  RETURN文が実行された時点でのESの値が関数の値として返される。
      RETURN
      END
```

（参考）C

```
#include <stdio.h>
#include <math.h>
```

```
/*========  関数プロトタイプ  ========*/

/*  プログラムの中で別に作成した関数を参照するときは、
    関数プロトタイプを作成する。
    関数プロトタイプは、include 文と同様に、
    ソースファイルの最初にまとめて記述する。  */
float f_es( float t );

/*========  飽和水蒸気圧を計算するための関数  ========*/

/*  関数の型は浮動小数点(float)型、関数名は f_es とする。
    引数は t(float 型)である。  */
float f_es( float t )
{

/*  変数を宣言する。  */
  float es;

/*  飽和水蒸気圧を計算する。
    exp は指数関数である。  */
  es = 611. * exp (17.27 * (t-273.16) / (t-35.86));

/*  関数は return 文で終了する。
    変数 es の値が関数 f_es の値として返される。  */
  return es;

}

/*========  main 関数  ========*/

/*  プログラムは main 関数から実行される。  */
int main( void )
{

/*  変数を宣言する。  */
  float t;

/*  入力を求めるメッセージを標準出力に書き出す。
    "¥n"は改行を表している。  */
  printf( "Temperature [deg.C]?¥n" );
```

```
/*  関数 scanf で標準入力から値を読みこむ。
    関数から値を受け取るときは変数名の前に&をつける。  */
  scanf( "%f", &t );

/*  t を絶対温度に変換する。  */
  t = t + 273.15;

/*  結果を標準出力に書き出す。
    関数 f_es に引数として温度 t の値を与えると、
    飽和水蒸気圧の値が返ってくる。
    飽和水蒸気圧の単位は Pa なので、hPa に変換するために 0.01 倍する。  */
  printf( "e_s [hPa] = %f¥n", 0.01*f_es( t ) );

  return 0;

}
```

実行例：

```
/home/snaoki> f77 prog06_2.f
/home/snaoki> ./a.out
 Temperature [deg.C]?
15.
 e_s [hPa] = 17.0480556
```

次に、行列とベクトルの積を計算するプログラムを作成します。ここでは次のような計算を考えます。

$$\begin{pmatrix} 0.7 & -0.7 & 0.0 \\ 0.7 & 0.7 & 0.0 \\ 0.0 & 0.0 & -1.0 \end{pmatrix} \begin{pmatrix} 2.0 \\ 1.0 \\ -1.0 \end{pmatrix} = \begin{pmatrix} 0.7 \\ 2.1 \\ 1.0 \end{pmatrix}$$

行列A　　ベクトルb　　ベクトルc

まず、構造化をしないで書いてみます。

プログラム：

FORTRAN

```
C  PARAMETER文で定数IMAXを宣言する。
      PARAMETER (IMAX=3)

C  配列A、B、Cを宣言する。
C  配列AはIMAX行IMAX列の行列に対応する。
```

```
C  また、配列Bと配列Cは要素数IMAXのベクトルに対応する。
      REAL A(IMAX,IMAX),B(IMAX),C(IMAX)

C  DATA文で配列A、Bを初期化する。
C  2次元配列の場合、
C  A(1,1),A(2,1),...,A(IMAX,1),A(1,2),A(2,2),...
C  の順である点に注意する。
      DATA A / 0.7, 0.7, 0.0,
     +        -0.7, 0.7, 0.0,
     +         0.0, 0.0,-1.0/
      DATA B / 2.0, 1.0,-1.0/

C  行列Aとベクトルbの積を計算する。

C  ここからDOループ11が始まる。
      DO 11 I=1,IMAX

C  C(I)にゼロを代入する。
        C(I) = 0.

C  ここからDOループ12が始まる。
        DO 12 J=1,IMAX

C  行列Aの(I,J)成分とベクトルBの第J成分の積を、
C  C(I)に加える。
          C(I) = C(I) + A(I,J) * B(J)

C  ここでDOループ12が終了する。
 12     CONTINUE

C  ここでDOループ11が終了する。
 11   CONTINUE

C  結果を書き出す。

C  ここからDOループ21が始まる。
      DO 21 I=1,IMAX

C  C(I)の値を標準出力に書き出す。
        WRITE(6,*) 'c[',I,']=',C(I)
```

```
C  ここでDOループ21が終了する。
 21   CONTINUE

      STOP
      END
```

（参考）C

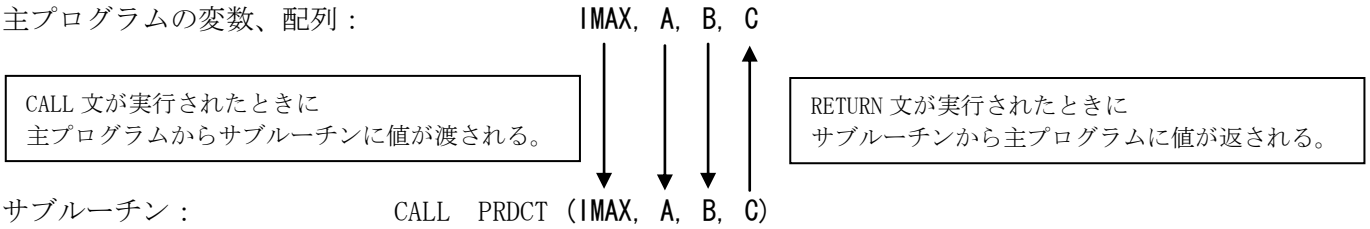
```
#include <stdio.h>

int main( void )
{

/*  変数 imax を宣言する。  */
  int imax=3, i, j;

/*  配列 a、b、c を宣言する。
    配列 a は imax 行 imax 列の行列に対応する。
    また、配列 b と配列 c は要素数 imax のベクトルに対応する。  */
  float a[3][3] = { 0.7,-0.7, 0.0,
                    0.7, 0.7, 0.0,
                    0.0, 0.0,-1.0 };
  float b[3] = { 2.0, 1.0,-1.0 };
  float c[3];

/*  行列 a とベクトル b の積を計算する。  */

/*  ここから for ループが始まる。  */
  for (i=0; i<=imax-1; i++)
  {

/*  c[i]にゼロを代入する。  */
    c[i] = 0.;

/*  ここから for ループが始まる。  */
    for (j=0; j<=imax-1; j++)
    {

/*  行列 a の(i,j)成分とベクトル b の第 j 成分の積を、
    c[i]に加える。  */
```

```
      c[i] = c[i] + a[i][j] * b[j];

/*  ここで for ループが終了する。  */
    }

/*  ここで for ループが終了する。  */
  }

/*  結果を書き出す。  */

/*  ここから for ループが始まる。  */
  for (i=0; i<=imax-1; i++)
  {

/*  c[i]の値を標準出力に書き出す。  */
    printf( "c[%d] = %f¥n", i+1, c[i] );

/*  ここで for ループが終了する。  */
  }

  return 0;

}
```

実行例：

```
/home/snaoki> f77 prog06_3.f
/home/snaoki> ./a.out
 c[ 1]=  0.699999988
 c[ 2]=  2.0999999
 c[ 3]=  1.
```

上のプログラムで、行列とベクトルの積を計算する部分を、サブルーチンにまとめてみます。FORTRAN の場合、関数とサブルーチンには区別があります。関数はそれ自体がひとつの値を返しますが、サブルーチンは、主プログラムから CALL 文を使って呼び出され、引数を通してデータの受け渡しを行います。複数の変数や配列を受け渡すこともできます。今回は配列の受け渡しをするので、サブルーチンを用いることにします。SUBROUTINE で始まる部分がサブルーチンです。サブルーチン名は PRDCT としています。



※実際にはソースコード上では下矢印と上矢印の区別はなく、CALL 文が実行されると引数に書かれているすべての変数、配列の値が主プログラムからサブルーチンに渡され、RETURN 文が実行されると引数に書かれているすべての変数、配列の値がサブルーチンから主プログラムに返されます。

C の場合、サブルーチンというものはなく、副プログラムはすべて関数という形で書きます。関数であっても、引数の部分でデータの受け渡しをすることができます。形式上、関数ですので、それ自体が値を返します。特に必要がない場合は、int 型とし、エラーがない場合には 0 を返すように書くのが普通です。

プログラム：

FORTRAN

```
C  PARAMETER文で定数IMAXを宣言する。
      PARAMETER (IMAX=3)

C  配列A、B、Cを宣言する。
C  配列AはIMAX行IMAX列の行列に対応する。
C  また配列Bと配列Cは要素数IMAXのベクトルに対応する。
      REAL A(IMAX,IMAX),B(IMAX),C(IMAX)

C  DATA文で配列A、Bを初期化する。
C  2次元配列の場合、
C  A(1,1),A(2,1),...,A(IMAX,1),A(1,2),A(2,2),...
C  の順である点に注意する。
      DATA A / 0.7, 0.7, 0.0,
     +        -0.7, 0.7, 0.0,
     +         0.0, 0.0,-1.0/
      DATA B / 2.0, 1.0,-1.0/

C  行列Aとベクトルの積を計算する。
      CALL PRDCT (IMAX,A,B,C)

C  結果を書き出す。

C  ここからDOループ21が始まる。
      DO 21 I=1,IMAX
```

```
C  C(I)の値を標準出力に書き出す。
        WRITE(6,*) 'c[',I,']=',C(I)

C  ここでDOループ21が終了する。
 21   CONTINUE

      STOP
      END

C  行列Aとベクトルbの積を計算する。
      SUBROUTINE PRDCT (IMAX,A,B,C)
      REAL A(IMAX,IMAX),B(IMAX),C(IMAX)

C  ここからDOループ11が始まる。
      DO 11 I=1,IMAX

C  C(I)にゼロを代入する。
        C(I) = 0.

C  ここからDOループ12が始まる。
        DO 12 J=1,IMAX

C  行列Aの(I,J)成分とベクトルBの第J成分の積を、
C  C(I)に加える。
          C(I) = C(I) + A(I,J) * B(J)

C  ここでDOループ12が終了する。
 12     CONTINUE

C  ここでDOループ11が終了する。
 11   CONTINUE

      RETURN
      END
```

（参考）C

```
#include <stdio.h>

/*========  関数プロトタイプ  ========*/
```

```
/*  プログラムの中で別に作成した関数を参照するときは、
    関数プロトタイプを作成する。
    関数プロトタイプは、include 文と同様に、
    ソースファイルの最初にまとめて記述する。  */
int product( int imax, float a[imax][imax], float b[imax], float c[imax] );

/*========  行列 a とベクトル b の積を計算するための関数  ========*/

/*  関数の種類は整数(int)型、関数名は product とする。
    引数は imax(int 型)、a、b、c(float 型配列のポインタ)である。  */
int product( int imax, float a[imax][imax], float b[imax], float c[imax] )
{

/*  変数を宣言する。  */
  int i, j;

/*  ここから for ループが始まる。  */
  for (i=0; i<=imax-1; i++)
  {

/*  c[i]にゼロを代入する。  */
    c[i] = 0.;

/*  ここから for ループが始まる。  */
    for (j=0; j<=imax-1; j++)
    {

/*  行列 a の(i,j)成分とベクトル b の第 j 成分の積を、
    c[i]に加える。  */
      c[i] = c[i] + a[i][j] * b[j];

/*  ここで for ループが終了する。  */
    }

/*  ここで for ループが終了する。  */
  }

  return 0;
```

```
}

/*========  main 関数  ========*/

/*  プログラムは main 関数から実行される。  */
int main( void )
{

/*  変数 imax を宣言する。  */
  int imax=3, i, status;

/*  配列 a、b、c を宣言する。
    配列 a は imax 行 imax 列の行列に対応する。
    また、配列 b と配列 c は要素数 imax のベクトルに対応する。  */
  float a[3][3] = { 0.7,-0.7, 0.0,
                    0.7, 0.7, 0.0,
                    0.0, 0.0,-1.0 };
  float b[3] = { 2.0, 1.0,-1.0 };
  float c[3];

/*  行列 a とベクトル b の積を計算する。  */
  status = product( imax, a, b, c );

/*  結果を書き出す。  */

/*  ここから for ループが始まる。  */
  for (i=0; i<=imax-1; i++)
  {

/*  c[i]の値を標準出力に書き出す。  */
    printf( "c[%d] = %f¥n", i+1, c[i] );

/*  ここで for ループが終了する。  */
  }

  return 0;

}
```

実行例：

```
/home/snaoki> f77 prog06_4.f
/home/snaoki> ./a.out
 c[ 1]=  0.699999988
 c[ 2]=  2.0999999
 c[ 3]=  1.
```

課題６：ふたつのベクトル（x, y, z）と（x', y', z'）の各成分を標準入力から入力し、内積と外積を計算して標準出力に出力するプログラムを作成せよ（report06.f[.c]）。ただし、

- ✓ FORTRAN の場合、内積の計算は関数、外積の計算はサブルーチンとしてそれぞれ副プログラムにまとめ、主プログラムから参照する形でプログラムを作成せよ。
- ✓ C の場合、内積と外積の計算をそれぞれ関数にまとめ、主プログラムから参照する形でプログラムを作成せよ。内積の計算結果は関数の戻り値として、外積の計算結果は引数に指定した配列として受け渡すようにせよ。

今回は３次元空間での計算を前提とするので、配列の大きさは、IMAX のような変数で与えるのではなく、はじめから明示的に３と指定してよい。

- ☞ 「x'」を出力したいときは「"x'"」とすればよい。
- ☞ C の場合、主プログラムと副プログラムとの間でベクトルや行列を引数として受け渡すときは、各成分をひとつずつスカラーの変数として受け渡すのではなく、配列としてまとめて受け渡す必要がある。