情報処理概論

第１１回　サブルーチン

情報基盤研究開発センター　谷本 輝夫

今回の内容

- 前回の課題 プログラム例
- サブルーチン

得点、合計点、平均点を表形式出力 プログラム例（1/3）

```
program Score_management
  implicit none
  integer :: i, j, total, number, kamoku
  integer, dimension(:,:), allocatable :: score
  integer, dimension(:), allocatable :: k_total
  intrinsic dble

  open(10, file='score.dat')
  read(10, *) number, kamoku

  allocate(score(number, kamoku))
  allocate(k_total(kamoku))

  ! Input score data
  do  i = 1, number
     read(10, *) score(i, 1:kamoku)
  end do

  close(10)
```

得点、合計点、平均点を表形式出力 プログラム例（2/3）

```
  ! Calculate total and average
    k_total = 0
  do  i = 1, number
     write(*,'(A3,I3,2X)', advance="NO") "No.",i
       total = 0
     do j = 1, kamoku
        total = total + score(i, j)
        write(*,'(I4,3X)', advance="NO") score(i,j)
        k_total(j) = k_total(j) + score(i,j)
     end do
     write(*,'(I4,3X)', advance="NO") total
     write(*,'(F6.1,1X)') total/dble(kamoku)
  end do

  write(*,'(A7,1X)', advance="NO") "Average"
  do j = 1, kamoku
     write(*,'(F6.1,1X)', advance="NO") k_total(j)/dble(number)
  end do
  write(*,*)
  stop
end program Score_management
```

得点、合計点、平均点を表形式出力 プログラム例（2/3）

```
$ cat score.dat
5 3
10 20 30
20 30 40
30 40 50
40 50 60
50 60 70
$ gfortran -o score score.f90
$ ./score
No.   1     10      20      30       60      20.0
No.   2     20      30      40       90      30.0
No.   3     30      40      50      120      40.0
No.   4     40      50      60      150      50.0
No.   5     50      60      70      180      60.0
Average    30.0    40.0    50.0
$
```

方程式を数値計算で解くプログラム
プログラム例（1/3）

```
program Kyukon
  implicit none
  real(8) :: a, b, c
  real(8), external :: f

  read(*, *) a, b
  do while (abs(a-b) > 0.000001)
    c = (a+b)/2
    if ((f(a) * f(c)) > 0) then
      a = c
    else
      b = c
    end if
    write(*,*) c
  enddo
  write(*,*) c

  stop
end program
```

```
program Kyukon
  implicit none
  real(8) :: a, b, c
  real(8), external :: f

  read(*, *) a, b
  do
    if (abs(a-b) <= 0.000001) then
      exit
    endif

    c = (a+b)/2
    if ((f(a) * f(c)) > 0) then
      a = c
    else
      b = c
    end if
    write(*,*) c
  enddo
  write(*,*) c

  stop
end program
```

方程式を数値計算で解くプログラム
プログラム例（2/3）

```
function f(x)
  implicit none
  real(8), intent(IN) :: x
  real(8) :: f

  f = x**3 + x -1
end function
```

方程式を数値計算で解くプログラム プログラム例（3/3）

```
$ ./enshu10-1
0
1
  0.50000000000000000
  0.75000000000000000
  0.62500000000000000
  0.68750000000000000
  0.65625000000000000
  0.67187500000000000
  0.67968750000000000
  0.68359375000000000
  0.68164062500000000
  0.68261718750000000
  0.68212890625000000
  0.68237304687500000
  0.68225097656250000
  0.68231201171875000
  0.68234252929687500
  0.68232727050781250
  0.68233489990234375
  0.68233108520507812
  0.68232917785644531
  0.68232822418212891
  0.68232822418212891
```

今回の内容

- 前回の課題 プログラム例
- サブルーチン

“サブルーチン”=プログラムの部品化

```
program temperature
  ...
   do i = 1, n
    do j = 1, n
      if (plate(i, j) >= 80) then
        write(*,'(a)',advance='NO') '#’
        ...
      end if
    end do
    write(*, *)
  end do
  write(*, *) 'Push Enter key'
  read(*, *)  ! wait for the Enter key
  ...
end program
```

```
program temperature
  ...
   call show(n, plate)
  ...
end program
```

サブルーチンの呼び出し

```
subroutine show(n, a)
implicit none
integer, intent(IN) :: n
real(8), dimension(n, n), intent(IN) :: a
integer :: i, j
  do i = 1, n
    do j = 1, n
      if (a(i, j) >= 80) then
        write(*, '(a)',advance='NO') '#'
        ...
      end if
    end do
    write(*, *)
  end do
  write(*, *) 'Push Enter key'
  read(*, *)
end subroutine
```

サブルーチン本体

サブルーチンの効果

- 何度も出てくる処理を一つにまとめることができる
- プログラム全体の構造がわかりやすくなる
- 一度作ったサブルーチンを、他のプログラムにも流用できる

主プログラムと副プログラム

- 主プログラム
  ＝　program ～ end program の部分
  - プログラムの中に必ず一つ存在
  - 実行は必ず主プログラムから開始される

- 副プログラム
  ＝　サブルーチン　もしくは　関数
  - プログラムの中で何個でも定義可能
  - 主プログラムや他の副プログラムから呼び出される

サブルーチンの定義

- 名前と引数（後述）を付けて定義
- 主プログラムの外側に記述する

```
subroutine サブルーチン名(引数)
  implicit none
  引数の宣言
   サブルーチン内で用いる変数や関数の宣言

  ... 処理 ...

end subroutine
```

- 例

```
subroutine input_data(file, n, a)
  implicit none
  integer, intent(IN) :: file, n
  integer, dimension(n), intent(OUT) :: a
  integer :: i

  do i = 1, n
    read(file, *) a(i)
  end do
end subroutine
```

サブルーチンの呼び出し

- サブルーチンの定義に合わせて引数を指定

```
call サブルーチン名(引数)
```

- 例

```
  ...
call input_data(10, number1, english)
...
```

引数（ひきすう）

- 副プログラムを呼び出す側と呼び出される側の間でデータを交換するための変数や値

```
call input_data(10, number1, english)
call input_data(11, number2, math)
```

呼び出し元の引数

```
subroutine input_data(file, n, a)
  implicit none
  integer, intent(IN) :: file, n
  integer, dimension(n), intent(OUT) :: a
  integer :: i

  do i = 1, n
    read(file, *) a(i)
  end do
end subroutine
```

サブルーチンの引数

呼び出し元とサブルーチンで引数の順番を合わせる

引数の特性の指定

- サブルーチンで引数の特性を指定できる
  - intent(IN)　　呼び出し元の値を参照
  - intent(OUT)　　呼び出し元に値を渡す
  - intent(INOUT)　呼び出し元の値を参照し、書き換え
- 省略可能だが、指定することにより、間違った引数の使い方をするとエラーで通知してもらえる

```
subroutine input_data(file, n, a)
  implicit none
  integer, intent(IN) :: file, n
  integer, dimension(n), intent(OUT) :: a
  integer :: i
  do i = 1, n
    read(file, *) a(i)
  end do
end subroutine
```

サブルーチン内で値が変わらない

サブルーチン内で値が書き換わる
＝　呼び出し元に渡される

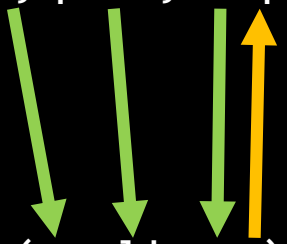
サブルーチンと引数の関係：例 1

```
program temperature
  ...
   call show(n, plate)
  ...
end program
  ...
```

```
subroutine show(n, a)
implicit none
integer, intent(IN) :: n
real(8), dimension(n, n), intent(IN) :: a
integer :: i, j
  do i = 1, n
    do j = 1, n
      if (a(i, j) >= 80) then
        write(*, '(a)',advance='NO') '#'
        ...
      end if
    end do
    write(*, *)
  end do
  write(*, *) 'Push Enter key'
  read(*, *)
end subroutine
```

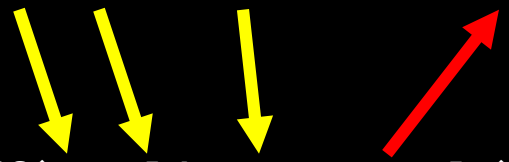
サブルーチンと引数の関係：例２

```
program temperature
  ...
    call step(n, plate, newplate)
  ...
end program

 ...

subroutine step(n, old, new)
implicit none
integer, intent(IN) :: n
real(8), dimension(n, n), intent(IN) :: old
real(8), dimension(n, n), intent(INOUT) :: new
integer :: i, j

  ! update 1 step
  do i = 2, n-1
    do j = 2, n-1
      new(i, j) = (old(i-1, j) + old(i+1, j) &
                 + old(i, j-1) + old(i, j+1)) / 4D0
    end do
  end do
end subroutine
```



サブルーチンと引数の関係：例３

```
program temperature
  ...
    call maxdiff(n, plate, newplate, check)
  ...
end program
 ...
subroutine maxdiff(n, old, new, result)
implicit none
integer, intent(IN) :: n
real(8), dimension(n, n), intent(IN) :: old, new
real(8), intent(OUT) :: result
real(8) :: diff
integer :: i, j
  result = 0d0
  do i = 2, n-1
    do j = 2, n-1
      diff = abs(new(i, j) - old(i, j))
      if (result < diff) then
        result = diff
      end if
    end do
  end do
end subroutine
```



変数の有効範囲

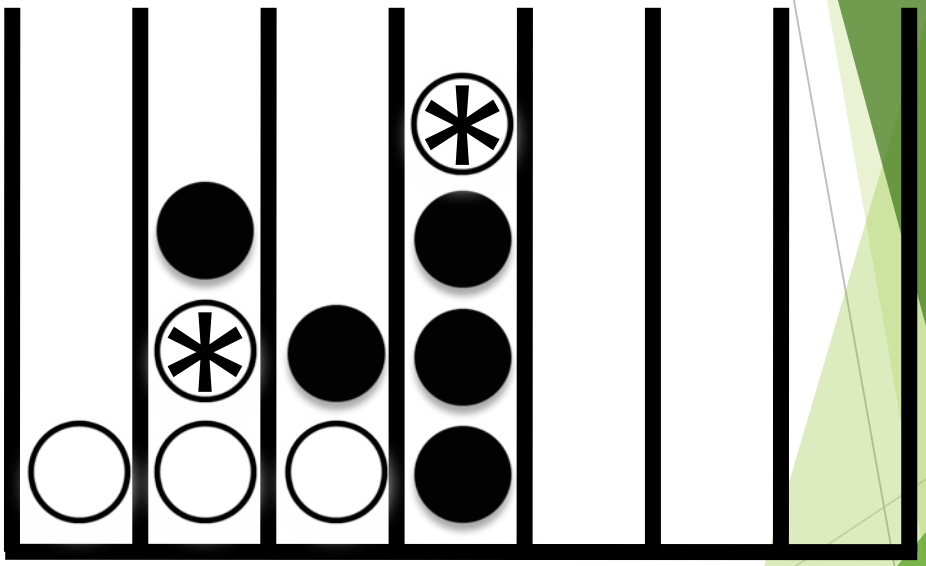
- 副プログラムや主プログラムで宣言された変数は、その副プログラムや主プログラムの中でのみ有効
  - 同じ名前の変数でもサブルーチンが違えば別の変数

```
subroutine a ( ... )
implicit none
...
integer :: x, y, z
   ...
end subroutine

subroutine b ( ... )
implicit none
...
integer :: x, y, z
   ...
end subroutine
```

サブルーチン a の変数 x, y, z と
サブルーチン b の変数 x, y, z は
無関係

演習11-1

- 次ページの主プログラムから呼び出すための立体四目並べ（商品名「ハムレット」）の盤面表示サブルーチンを作成する
- 立体四目並べ
- 上から交互に玉を落としてゆき、先に縦、横、斜めのいずれかの方向で四目並べれば勝ち



- 今回のプログラムでは　＊　と o で玉を表現

演習用の主プログラム

```
program four
implicit none
integer, parameter :: m=7, n=6
character(len=1), dimension(m, n) :: board

  board = ' '
  board(1, 1) = '*'
  board(4, 1) = 'o'
  board(2, 1) = '*'
  board(4, 2) = 'o'
  board(2, 2) = '*'
  board(4, 3) = 'o'
  board(4, 4) = '*'
  board(2, 3) = 'o'
  board(3, 1) = '*'
  board(3, 2) = 'o'
  call show(m, n, board)
stop
end program
```

表示例

```
  1  2  3  4  5  6  7
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][*][ ][ ][ ]
[ ][o][ ][o][ ][ ][ ]
[ ][*][o][o][ ][ ][ ]
[*][*][*][o][ ][ ][ ]
```

列番号や、列の区切りも表示する

文字列について

- 文字列：　0個以上の文字の並び
- 文字列変数の宣言

```
character(len=長さ) :: 変数名
```

  - 整数や実数と同様に操作できる
    - 値の代入

```
a = '*'
```

    - 表示

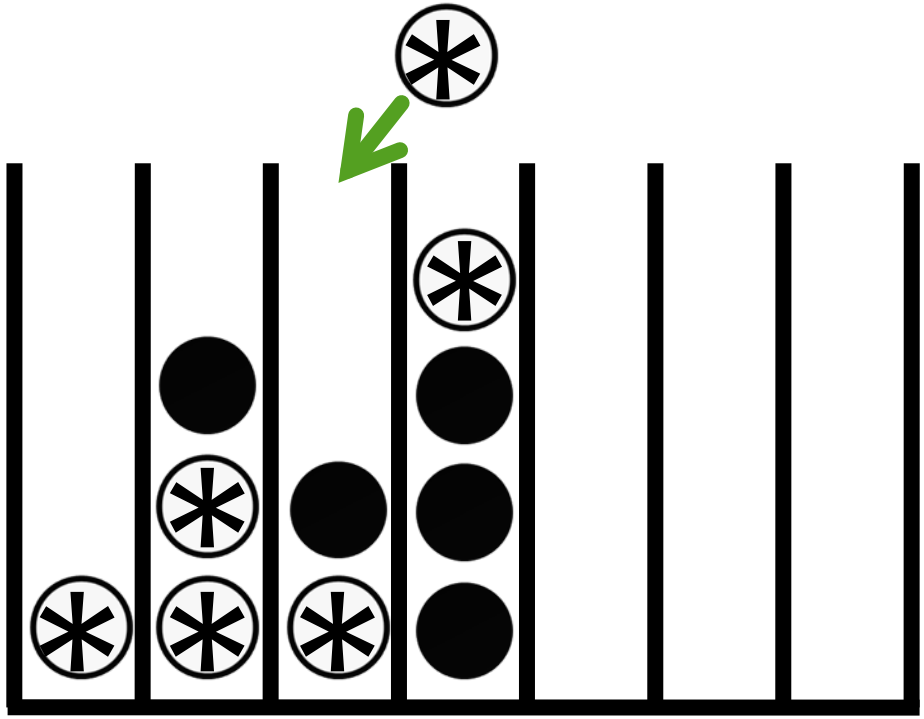
```
write(*, '(a)') b
```

    - 比較

```
if (c /= 'x') then
```

演習11-2

- 次ページの立体四目並べの主プログラムから呼び出す、入力サブルーチン drop を作成する



立体四目並べの主プログラム

```
program four
implicit none
integer, parameter :: m=7, n=6
character(len=1), dimension(m, n) :: board
integer :: step, side

  board = ' '
  call show(m, n, board) ! Initial status

  do step = 1, m*n/2
    do side = 1, 2
      ! Drop ball to the specified column
      call drop(m, n, board, side)
      call show(m, n, board)
    end do
  end do
stop
end program
```

dropサブルーチンの仕様

- 引数：　４個
  - 盤の幅（＝列数）　… 整数
  - 盤の高さ（＝行数）　… 整数
  - 盤の配列　… 文字列（len=1)の２次元配列
  - 手番　… 整数（１：先攻 (*) 、 ２：後攻 (o)）
- 玉を落とす列の番号を入力させ、
  それに応じて盤の配列を更新する
- 間違った入力がされた場合、
  メッセージを表示して再度入力
  - 盤の範囲から外れた列番号
    - ただし、0 が入力されると終了することにする
  - もう玉を入れることのできない列の番号

表示例

```
   1  2  3  4  5  6  7
 [ ][ ][ ][ ][ ][ ][ ]
 [ ][ ][ ][ ][ ][ ][ ]
 [ ][ ][ ][ ][ ][ ][ ]
 [ ][ ][*][ ][ ][ ][ ]
 [ ][ ][o][*][ ][ ][ ]
 [*][*][o][o][o][*][ ]
o : Drop where? (0 = Exit)
```

どちらの手番かを表示する

エラーの表示例

```
  1  2  3  4  5  6  7
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[*][ ][*][o][o][ ][ ]
* : Drop where? (0 = Exit)8
  Out of range!!
* : Drop where? (0 = Exit)
```

```
  1  2  3  4  5  6  7
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ]
[ ][ ][o][*][ ][ ][ ]
[ ][*][o][*][ ][ ][ ]
[ ][o][*][o][ ][ ][ ]
[ ][*][o][*][ ][ ][ ]
[ ][*][o][o][ ][ ][ ]
[*][o][*][o][*][ ][ ]
o : Drop where? (0 = Exit)3
 This column is full!!
o : Drop where? (0 = Exit)
```

間違った場所に置こうとするとメッセージを表示して再入力

出来た人は

- 次回作成予定の、
  勝敗判定サブルーチンの設計を始める
- 縦、横、斜めのいずれかの方向で四目揃ったかどうかを判定
- 主プログラムや　dropサブルーチンは、
  必要に応じて適宜変更する

進め方がわからない人は

- まず、主プログラムを入力
  - 中身をじっくり読んで、理解する
- 次に show サブルーチンや drop サブルーチンの枠組みを入力
  - 枠組み： subroutine, end subroutineの行、および各引数の宣言
  - 例）showサブルーチンの枠組み

```
subroutine show(m, n, board)
implicit none
integer, intent(IN) :: m, n
character(len=1), dimension(m, n), intent(IN) :: board

end subroutine
```

ここまでで、とりあえずコンパイル、実行してみる

サブルーチンの中身の作成

- まず、サブルーチンでやるべきことを列挙し、
  分かる部分から段階的に実装して、
  それぞれが正しく動く事を確認していく。
  例えば…

  第１段階　手番の番号（１or２）に応じて記号（＊ or ｏ）
  を表示する

  第２段階　列番号の入力を促すメッセージ表示の write
  および入力のための read 文を追加

  第３段階　入力された列番号が
  0 なら終了
  盤の範囲外なら 'Out of range!!' の表示

  第４段階　正しい入力がされるまで繰り返す

  ・・・

センチネル（番兵）の利用

第４段階　正しい入力がされるまで繰り返す

- 繰り返す → ループ文
- 終了条件 → 正しい入力
- 正しい入力とは何だろうか？
  - 0ではない
  - 盤の範囲内（１～７）
  - その列が一杯ではない

→ 判定条件が沢山あって大変

- その他のループ制御
  - exit
  - cycle
  - stop

センチネルを利用

```
flag = 0
do while (flag == 0)
   :
   if ( . . .)
      flag = 1
   endif
   :
end do
```