情報処理概論

第５回　制御文

情報基盤研究開発センター　谷本 輝夫

先週の課題の解答例

```
program area_of_circle
  implicit none
  real(8) :: r, pi
  real(8) :: Base, Height
  intrinsic atan, sin, cos

  pi = 4D0 * atan(1D0)
  write(*, *) 'Hankei?'
  read(*, *) r
  Base = r * cos(pi * 30 / 180) * 2
  Height = r * sin(pi * 30 / 180) + r
  write(*, '(A10 F5.2)') 'Menseki :', Base * Height / 2
stop
end program
```

今日の予習

```
program sample1
implicit none
integer :: i, j, n
intrinsic  dble

  ! Input N
  write(*, *) 'Enter N :'
  read(*, *) n

  ! Check the value of N
  if (n <= 0) then
    write(*, *) 'Error: N must be > 0'    ! Wrong value
  else
    ! Print out i * j for all  (1 <= i,j <= N )
    do i = 1, n
      do j = 1, n
        write(*, *) i, ' / ', j, ' = ', dble(i) / dble(j)
      end do
    end do
  end if
stop
end program
```

キーボードから入力した値を n に格納

n の値が0以下の場合

それ以外の場合

i = 1～n のそれぞれについて

j = 1～n のそれぞれについて

今回の内容

- 条件分岐
- 繰り返し
- 型を変換するための関数
- 読みやすいプログラム

条件分岐
if 文（もし～ならば）

- 例）降水確率が５０%以上なら ”傘を持っていきなさい” と表示

```
program pillar
  implicit none
  integer :: prob

  write(*, *) 'Probability of the rain: '
  read(*, *) prob

  if (prob >= 50) then
    write(*, *) "May be it will rain."
    write(*, *) "Don't forget to bring an umbrella!"
  end if

stop
end program
```

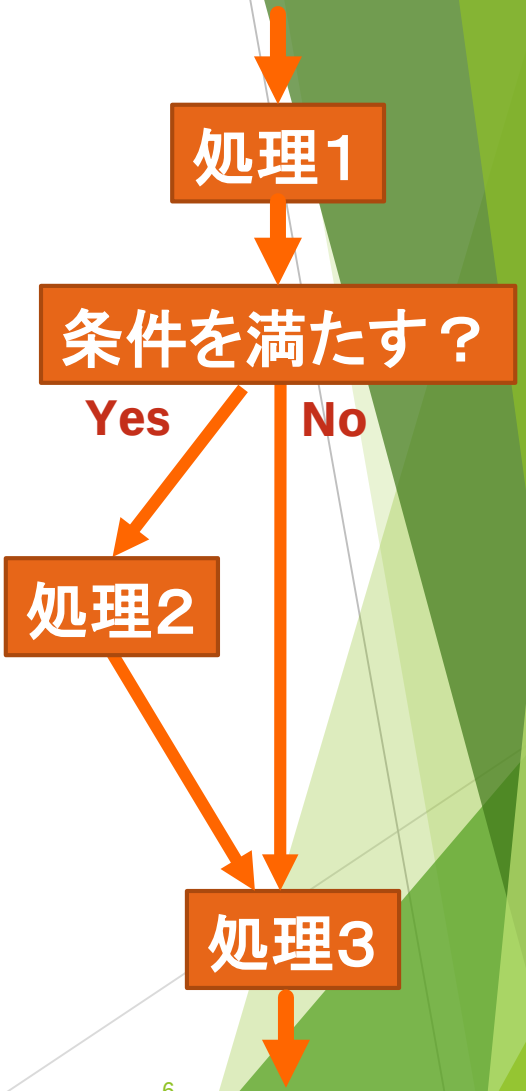
if 文の利用法（１）

- もし～ならば

```
・・・
処理１

if（条件）then
   処理２
end if

処理３
・・・
```



条件

- if 文で利用できる主な条件

新表記	旧表記	意味
>	.gt.	より大きい
<	.lt.	より小さい
>=	.ge.	以上
<=	.le.	以下
/=	.ne.	等しくない
==	.eq.	等しい

= が一つだけだと問題がある
どんな問題？

条件を満たさなかった場合の処理

- ５０%以上なら ”傘を持っていきなさい”
  さもなければ ”持っていかなくてもいいかも”

```
if (prob >= 50) then
  write(*, *) "May be it will rain."
  write(*, *) "Don't forget to bring an umbrella!"
else
  write(*, *) "May be it will not rain."
  write(*, *) "You may go without an umbrella."
end if
```

if 文の利用法（２）

- もし～ならば～さもなくば～

```
if（条件）then
   処理１
else
   処理２
end if
```

もっと細かく場合分け

```
if (prob >= 70) then
  write(*, *) "Don't forget to bring an umbrella!"
else
  if (prob >= 50) then
    write(*, *) "You should bring an umbrella."
  else
    if (prob >= 30) then
      write(*, *) "You may go without an umbrella."
    else
      write(*, *) "You don't need an umbrella."
    end if
  end if
end if
```

どちらも同じ意味

```
if (prob >= 70) then
  write(*, *) "Don't forget to bring an umbrella!"
else if (prob >= 50) then
  write(*, *) "You should bring an umbrella."
else if (prob >= 30) then
  write(*, *) "You may go without an umbrella."
else
  write(*, *) "You don't need an umbrella."
end if
```

if 文の利用法（３）

- もし(条件A)ならば～
  さもなくば，もし(条件B)ならば～
  さもなくば，もし(条件C)ならば～
  …
  さもなくば～

```
if (条件A) then
  条件Aを満たす場合の処理
else if (条件B) then
  条件Aを満たさず，条件Bを満たす場合の処理
else if (条件C) then
  条件Aも条件Bも満たさず，条件Cを満たす場合の処理

    ...

else
  どの条件も満たさない場合の処理
end if
```

複数の条件の組み合わせ：”または”と”かつ”

- 条件１　または　条件２

```
if （条件１ .or. 条件２） then
```

- 条件１　かつ　条件２

```
if （条件１ .and. 条件２） then
```

前後のピリオドを忘れずに！

.or. の利用例

- 降水確率が 0 未満 または 100 より大きい場合エラー

```
if ( prob  <  0 .or. prob  >  100 ) then
  write(*,*) "Error: Wrong probability ", prob
  stop
else if (prob  >=  70) then
  write(*,*) "Don't forget to bring an umbrella!"
else if (prob  >=  50) then
  write(*,*) "You should bring an umbrella."
else if (prob >= 30) then
  write(*,*) "You may go without an umbrella."
else
  write(*,*) "You don't need an umbrella."
end if
```

.and. の利用例

- 降水確率が 0 未満 または 100 より大きい場合エラー

```
if ( prob  >=  0 .and. prob  <=  100 ) then
  if (prob  >=  70) then
    write(*,*) "Don't forget to bring an umbrella!"
  else if (prob  >=  50) then
    write(*,*) "You should bring an umbrella."
  else if (prob  >=  30) then
    write(*,*) "You may go without an umbrella."
  else
    write(*,*) "You don't need an umbrella."
  end if
else
  write(*,*) "Error: Wrong probability ", prob
  stop
end if
```

前のページの .or. の利用例と比較してみる

もう一つの方法
select 文

- 前のページの例を select 文で置き換えた例：

```
select case(prob)
case(70:100)
  write(*, *) "Don't forget to bring an umbrella!"
case(50:69)
  write(*, *) "You should bring an umbrella."
case(30:49)
  write(*, *) "You may go without an umbrella."
case(0:29)
  write(*, *) "You don't need an umbrella."
case default
  write(*, *) "Error: Wrong probability ", prob
  stop
end select
```

前のページの例と比較してみる

select文の利用法

```
select case(式)
case(範囲１)
    式の値が範囲１に含まれる場合の処理
case(範囲２)
    式の値が範囲２に含まれる場合の処理
...
case default
  式の値がどの値にも含まれない場合の処理
end select
```

- "式" は整数型か文字型か論理型しか利用できない
  - 実数は不可
- case default：
  - どの値にも含まれない場合の処理　(省略可)

case で指定できる値

- ある一つの値のみ
  例)
  ```
  case (20)
  ```
  20 の場合
  ```
  case ('d')
  ```
  'd' の場合
- 数値や文字列の範囲
  例)
  ```
  case (0:10)
  ```
  0以上 10以下の場合
  ```
  case ('a':'e')
  ```
  'a', 'b', ..., 'e' のどれかの場合
  ```
  case (:50)
  ```
  50 以下の場合
  ```
  case ('n':)
  ```
  'n', 'o', ..., 'z' のどれかの場合
- 複数の値
  例)
  ```
  case (1, 5, 10)
  ```
  1, 5, 10 のどれかの場合
  ```
  case ('c', 't')
  ```
  'c', 't' のどれかの場合

今回の内容

- 条件分岐
- 繰り返し
- 型を変換するための関数
- 読みやすいプログラム

繰り返し処理の必要性

- 例）以下のように5個の整数をキーボードから入力して総和を計算するプログラムを作る

```
$ ./test
 Input data 1:
10
 Input data 2:
4
 Input data 3:
8
 Input data 4:
16
 Input data 5:
22
 Total is  60
```

方法１：同じ処理を5回書く

```
program sum
implicit none
  integer :: total, data

  total = 0  ! Initialize
  write(*, *)  'Input data 1:'
  read(*, *) data
  total = total + data
  write(*, *)  'Input data 2:'
  read(*, *) data
  total = total + data
  write(*, *)  'Input data 3:'
  read(*, *) data
  total = total + data
  write(*, *)  'Input data 4:'
  read(*, *) data
  total = total + data
  write(*, *)  'Input data 5:'
  read(*, *) data
  total = total + data

  write(*, *) 'Total is ', total

  stop
end program
```

- あまり現実的ではない
  - データの数が多くなるとプログラムの入力や書き換えが大変
  - データの数が変わる度にプログラムを変更しなければならない

方法２：同じ処理を繰り返す

- i を 1 から 5 まで 1つずつ増やしながら
  プログラムの一部を繰り返す

```
program sum
implicit none
  integer :: total, i, data

  total = 0  ! Initialize

  do i = 1, 5
    write(*, *)  'Input data ', i, ':'
    read(*, *) data
    total = total + data
  end do

  write(*, *) 'Total is ', total

  stop
end program
```

} i = 1, ..., 5 について，それぞれこの部分を繰り返し実行

実行の流れ

```
...
do i = 1, 5
  write(*, *) 'Input data ', i, ':'
  read(*, *) data
  total = total + data
end do
...
```

i=1

```
write(*, *) 'Input data ', 1, ':'
read(*, *) data
total = total + data
```

i=2

```
write(*, *) 'Input data ', 2, ':'
read(*, *) data
total = total + data
```

i=5

```
write(*, *) 'Input data ', 5, ':'
read(*, *) data
total = total + data
```

total の値は、どのように変化する？

do文の利用法

```
do 制御変数 = 開始値，終了値 ，増分
    繰り返す処理
end do
```

- 制御変数の値を 開始値 から 増分 ずつ
  増やしてゆき，終了値 より大きくなるまで繰り返す
  - 例）i を 0 から 2, 4, 6, …, 10 と増やしていく

```
do i = 0, 10, 2
```

- 増分を省略すると 1 が指定されたとみなされる
- 増分は負の値でも良い.
  - この場合， 制御変数を開始値から増分ずつ減らしてゆき，
    終了値より小さくなるまで繰り返す.
  - 例） i を 100から 95, 90, 85, … , 0 と減らしていく

```
do i = 100, 0, -5
```

開始値、終了値、増分は変数や数式でも構わない

- 例

```
program sum
  implicit none
  integer :: total, i, data, n

  total = 0  ! Initialize
  write(*, *) 'Input number of data :'
  read(*, *) n

  do i = 1, n
    write(*, *)  'Input data ', i, ':'
    read(*, *) data
    total = total + data
  end do

  write(*, *) 'Total is ', total

  stop
end program
```

1 から n まで

do文利用上の注意

- do文の繰り返し処理中，制御変数の値を変化させない
  - 途中で値が変化すると繰り返しが正常に終了しない
- 誤りの例:

```
do i = 1, 5
  write(*, *)  'Input data ', i, ':'
  read(*, *) i
  total = total + i
end do
```

どのように動作するか予想してみる

exit
do 文を途中で抜ける

- end do の次の処理に移る

```
do i = 1, 5
  write(*, *)  'Input data ', i, ':'
  read(*, *) data
  if (data < 0) then
    exit
  end if
  total = total + data
end do
次の処理
```

data に負の数が入力されたら
繰り返しから抜ける

- 注）多重のdo文の場合，一つ外側に抜ける
- 右の例では，exitすると i の次の繰り返しを継続

```
do i = 1, 5
  do j = 1, 10
    ...
      exit
    ...
  end do
end do
```

文の組み合わせ

- if文やdo文はいくつでも組み合わせて記述できる
  - 注意：doとend do, ifとend ifは入れ子構造

```
program sum
  implicit none
  integer :: total, i, data, n

  total = 0  ! Initialize
  write(*, *) 'Input number of data :'
  read(*, *) n
  if (n > 0) then
    do i = 1, n
      write(*, *)  'Input data ', i, ':'
      read(*, *) data
      total = total + data
    end do
  end if
  write(*, *) 'Total is ', total
  stop
end program
```

間違いの例

```
  if (n > 0) then
    do i = 1, n
      write(*, *) 'Input data ',i,':'
      read(*, *) data
      total = total + data
  end if
end do
```

どこが間違い？

もう一つの繰り返し処理：do while 文

- 条件を満たす間，繰り返し

```
program sum
implicit none
  integer :: total, data, i

  ! Initialize
  total = 0
  i = 1

  do while (total < 100)
      write(*, *)  'Input data ', i, ':'
      read(*, *) data
      total = total + data
      i = i + 1
  end do

  write(*, *) 'Total is ', total, ' , Number of items is ', i
  stop
end program
```

total が 100 に到達するまでデータ入力を続ける

do while 文の利用法

```
do while (条件)

    繰り返す処理

end do
```

- 条件を満たす間，繰り返し処理を継続
- 注意
  - 繰り返し処理の途中で条件判定が変わってもその回の処理は全て実行され，do whileに戻った際に継続か否かが判定される

```
do while (i <= 100)
  i = i + 1
  write (*, *) i
end do
```

101 は、表示されるか？

今回の内容

- 条件分岐
- 繰り返し
- 型を変換するための関数
- 読みやすいプログラム

型の自動変換

- 前々回の講義で紹介したように，整数型と実数型の演算が混在する場合，型の強い方に自動変換される

強さ: 倍精度実数＞単精度実数＞整数

- 以下は全て同じ結果

```
4D0 * 3D0 / 7D0
```

```
4.0 * 3D0 / 7.0
```

```
4 * 3D0 / 7
```

型の明示的な変換の必要性

- 型が混在するプログラムを記述する場合十分な注意が必要
- 以下は互いに結果が異なる

```
4D0 * (3D0 / 7D0)
```

```
4D0 * (3.0 / 7.0)
```

```
4D0 * (3 / 7)
```

必ず意図した結果が得られるように，
型変換関数を使って，計算式中の型を揃えた方が安心

型の変換：整数→実数
dble 関数

- 制御変数や番号に用いている整数値を実数の計算に利用する場合等

```
program to_real
  implicit none
  integer :: i, j
  intrinsic dble

  do i = 1, 10
    do j = 1, 10
      write(*, *) i, '/', j, ' = ', dble(i) / dble(j)
    end do
  end do

stop
end program
```

型の変換：実数→整数
int関数, ceiling関数, nint関数

- 小数点以下の扱いによって使い分け
  - 切り捨て　int
    - 例）`int(3.1)` → 3
  - 切り上げ　ceiling
    - 例）`ceiling(3.1)` → 4
  - 四捨五入　nint
    - 例）`nint(3.1)` → 3

今回の内容

- 条件分岐
- 繰り返し
- 型を変換するための関数
- 読みやすいプログラム

余白を上手に使う

プログラムの
構造に合わせて
字下げ

```
program sample1
implicit none
integer :: i, j, n

  write(*, *) 'Enter N :'
  read(*, *) n

  if (n <= 0) then
    write(*, *) 'Error: N must be > 0'
  else
    do i = 1, n
      do j = 1, n
        write(*, *) i, ' * ', j, ' = ', i * j
      end do
    end do
  end if

stop
end program
```

プログラムの
まとまりごとに
空行を挿入

字下げ

- プログラムの構造にあわせて字下げ
- 通常、if 文や do文の中の処理を1段字下げする
  →プログラムの相互関係がわかりやすい
- 字下げの有無による違い：

```
program sample1
implicit none
integer :: i, j, n

write(*, *) 'Enter N :'
read(*, *) n
if (n <= 0) then
write(*, *) 'Error: N must be > 0'
else
do i = 1, n
do j = 1, n
write(*, *) i, ' * ', j, ' = ', i * j
end do
end do
end if
stop
end program
```

```
program sample1
implicit none
integer :: i, j, n

  write(*, *) 'Enter N :'
  read(*, *) n
  if (n <= 0) then
    write(*, *) 'Error: N must be > 0'
  else
    do i = 1, n
      do j = 1, n
        write(*, *) i, ' * ', j, ' = ', i * j
      end do
    end do
  end if
stop
end program
```

注釈の利用

- Fortranのプログラムにおける各行について，感嘆符！以降は注釈として扱われる
  - 注釈：プログラムとしては何も意味を持たない，メモのようなもの
    - 自分が後で読むときの備忘録
    - 他人にプログラムを見せるときの解説文

```
! Input N
write(*, *) 'Enter N :'
read(*, *) n

! Check the value of N
if (n <= 0) then
  write(*, *) 'Error: N must be > 0'   ! Wrong value
else
  ! Print out i * j for all  (1 <= i,j <= N )
  do i = 1, n
```

今日の課題

- 予習のページにあるプログラムを入力し、動かす
- 条件分岐と繰り返しを使ったプログラムを自分で作成する
  - 入力した値までの掛け算表（掛け算九九）を出力
  - 素数を見つけて出力
  - トランプのカードを入力して、ポーカーの役を出力
  - 住宅ローン返済プラン作成
  - 割り勘計算プログラム
  - etc…