

Fortran 入門

はじめに

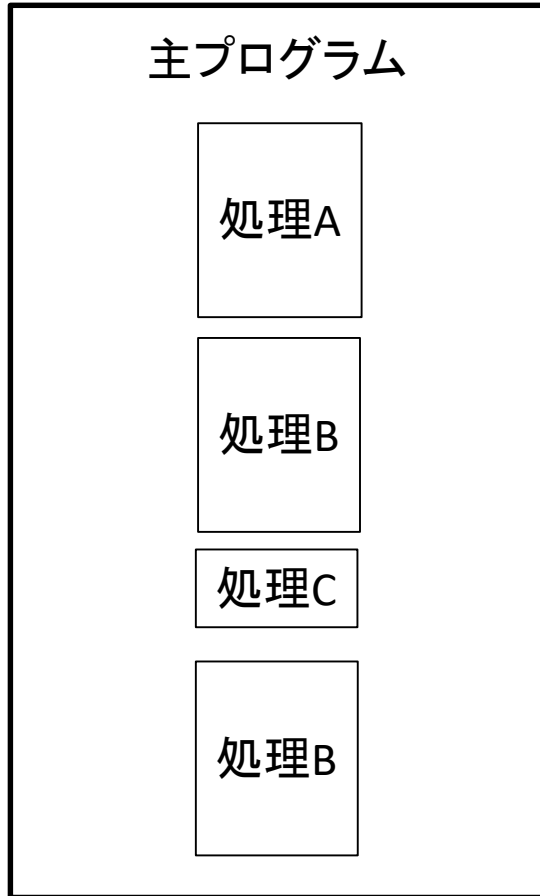
- 長いプログラムにおいて、ひとつの program 文にすべての処理を書くと困ったことが起こる。
 - プログラムの流れを把握しにくい
 - プログラムに間違いがあった時に間違った箇所を特定するのが難しい,
 - プログラム内のある処理を別のプログラムに使いたくても使えない,
 - エディタ(例えば emacs)でプログラムの編集箇所にとどり着くのに時間がかかる,
 - ...

サブルーチンと関数

- このような時のために Fortran では二つの仕組みが用意されている.
 - サブルーチン
 - 関数
- Fortran ではこれらをまとめて副プログラムと呼ぶ.
 - それに対して, program 文があるプログラムの単位を主プログラムと呼ぶ.
- 他のプログラミング言語でも, (比較的) 小さなプログラムの集まりによって大規模なプログラムを構成する方法が用意されている.

プログラムのイメージ 1-1

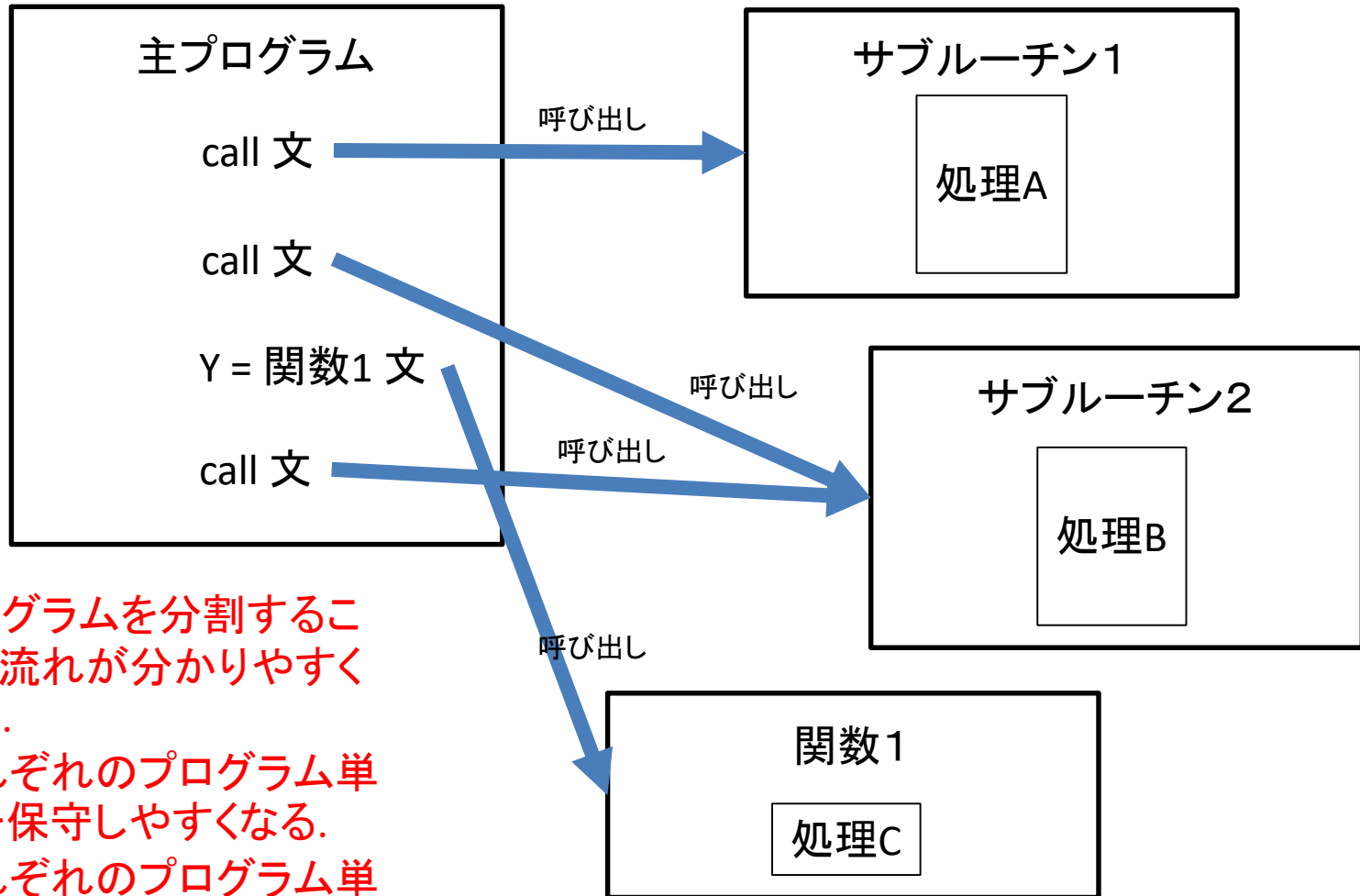
サブルーチンと関数を用いない場合



- サブルーチンや関数を使わない時
 - 主プログラム内にすべての処理を記述

プログラムのイメージ 1-2

サブルーチンと関数を用いる場合



- プログラムを分割することで流れが分かりやすくなる.
- それぞれのプログラム単位を保守しやすくなる.
- それぞれのプログラム単位は何度でも使える.

プログラムのイメージ 2-1

サブルーチンと関数を用いない場合

主プログラム1

処理A

処理B

処理C

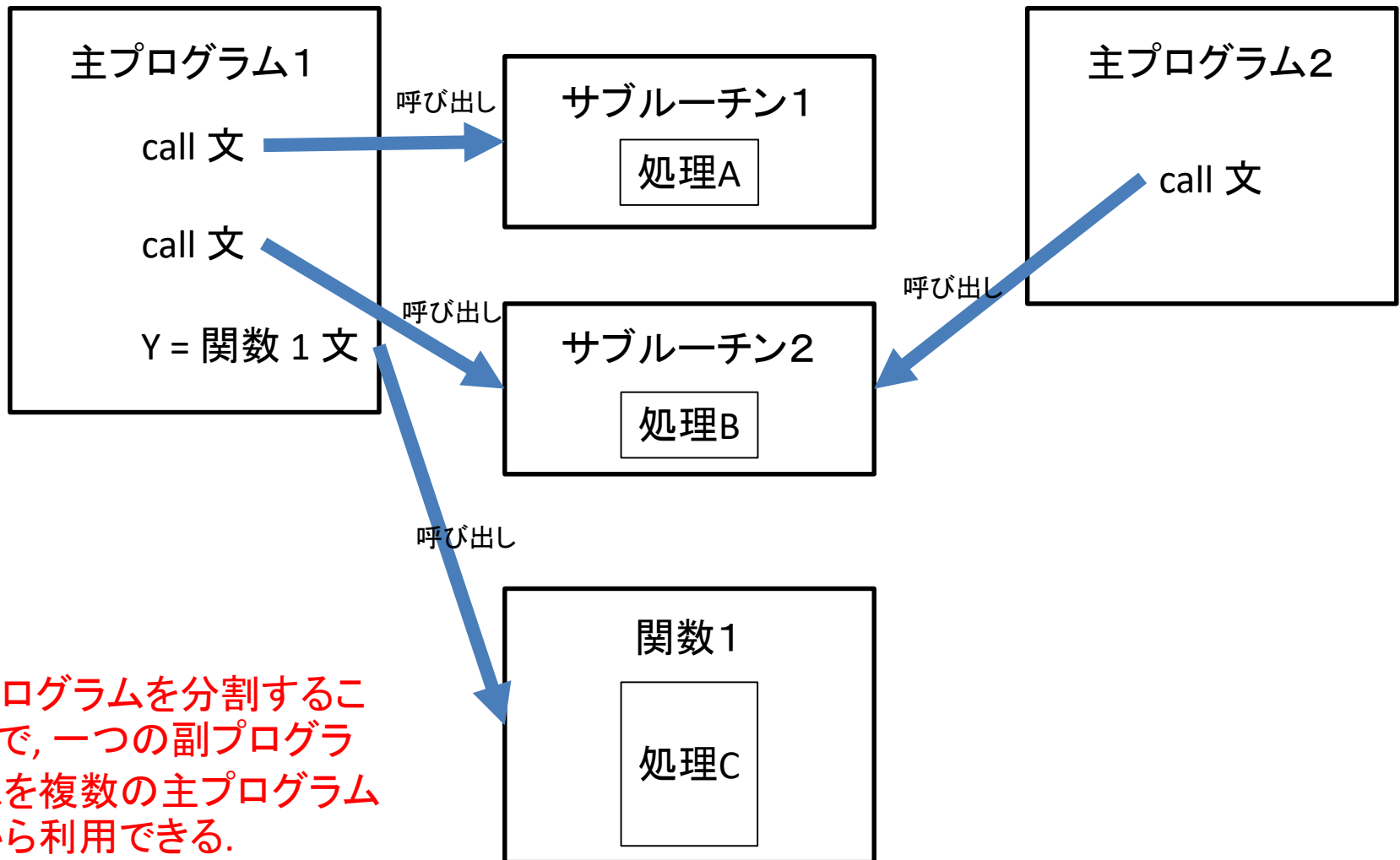
- 二つのプログラム
- 一部は同じ処理を行う
- サブルーチンや関数を使わない時
 - 主プログラム内にすべての処理を記述

主プログラム2

処理B

プログラムのイメージ 2-2

サブルーチンと関数を用いる場合



- プログラムを分割することで、一つの副プログラムを複数の主プログラムから利用できる。

プログラムの具体例

program 文だけで書くプログラム

1 から 10 までの和を計算するプログラム


```
program summation

implicit none

integer :: i
integer :: num

! 1 から 10 までの和の計算
num = 0           ! num の初期化
do i = 1, 10
  num = num + i
end do
write( 6, * ) num

end program summation
```



この部分を取り出してサブルーチン・関数にしてみる

プログラムの具体例

サブルーチンを使ったプログラム

1 から 10 までの和を計算するサブルーチン

```
program summationsub  
  
implicit none  
  
integer :: num  
  
! 1 から 10 までの和の計算  
call calcsun(10,num)  
write( 6, * ) num  
  
end program summationsub
```

呼び出し

```
! サブルーチンの定義  
subroutine calcsun(n,kazu)  
  
implicit none  
  
integer, intent(in) :: n  
integer, intent(out) :: kazu  
  
integer :: i  
  
kazu = 0           ! kazu の初期化  
do i = 1, n  
    kazu = kazu + i  
end do  
  
end subroutine calcsun
```

サブルーチンの最低限の決まり

- サブルーチンは主プログラム/副プログラムから call によって呼び出す.
- subroutine 文で始まり, end subroutine 文で終わる.
- program 文とは別にサブルーチン内で使う変数の宣言が必要.
- program 文とは別に implicit none が必要.
- program 文からの入力, program 文への出力は, 引数で行う.

! サブルーチンの定義

```
subroutine calcsun(n,kazu)
```

```
implicit none
```


引数

```
integer, intent(in) :: n
```

```
integer, intent(out) :: kazu
```

```
integer :: i
```

```
kazu = 0
```

! kazu の初期化

```
do i = 1, n
```

```
    kazu = kazu + i
```

```
end do
```

```
end subroutine calcsun
```

プログラムの具体例

関数を使ったプログラム

1 から 10 までの和を計算する関数

```
program summationsub  
  
  implicit none  
  
  integer :: num  
  
  ! 1 から 10 までの和の計算  
  num = calsumfunc(10)  
  write( 6, * ) num  
  
end program summationsub
```

呼び出し

```
! 関数の定義  
function calsumfunc(n) result(kazu)  
  
  implicit none  
  
  integer, intent(in) :: n  
  
  integer :: kazu  
  integer :: i  
  
  kazu = 0  
  do i = 1, n  
    kazu = kazu + i  
  end do  
  
end function calsumfunc
```

関数の最低限の決まり

- 関数は主プログラム/副プログラムから下のよう呼び出す.
 - $y = f(x1, x2)$
- function 文で始まり, end function 文で終わる.
- program 文とは別に関数内で使う変数の宣言が必要.
- program 文とは別に implicit none が必要.
- program 文からの入力は引数で行う.
- program 文への出力は, 戻り値で行う.

! 関数の定義

```
function calcsfunc(n) result(kazu)
```

```
implicit none
```

 
引数 戻り値

```
integer, intent(in) :: n
```

```
integer :: kazu
```

```
integer :: i
```

```
kazu = 0
```

```
do i = 1, n
```

```
  kazu = kazu + i
```

```
end do
```

```
end function calcsfunc
```

サブルーチンと関数の違い

- サブルーチン
 - 入力にも出力にも引数を用いる
 - 複数の変数を出力に用いることができる
- 関数
 - 入力には引数を用いるが, 出力には(普通は)戻り値を用いる
 - 実際には引数を出力に用いることもできる
 - 戻り値は一つのみ
 - 数式のようにプログラムを書くことができる

変数のスコープ

- 変数には, その変数が有効となる範囲がある.
 - 「スコープ」と呼ぶ.
- 変数は, 基本的には, 宣言したプログラム単位内のみで有効
 - 次ページ参照
 - どこでも有効な「グローバル変数」も存在するが, ここでは触れない.

変数のスコープ

1 から 10 までの和を計算するサブルーチン

```
program summationsub
```

```
implicit none
```

この変数 num は主プログラム内のみで有効

```
integer :: num
```

```
! 1 から 10 までの和の計算
```

呼び出し

```
call calcsun(10,num)
```

```
write( 6, * ) num
```

```
end program summationsub
```

主プログラムの変数 num の値は、
calcsun の kazu に渡されるが、
それらは別の変数扱い。

```
! サブルーチンの定義
```

```
subroutine calcsun(n,kazu)
```

```
implicit none
```

変数 n, kazu, i は calcsun 内のみで有効

```
integer, intent(in) :: n
```

```
integer, intent(out) :: kazu
```

```
integer :: i
```

```
kazu = 0
```

! kazu の初期化

```
do i = 1, n
```

```
    kazu = kazu + i
```

```
end do
```

```
end subroutine calcsun
```

複数ファイルに分けたプログラム

- 主プログラムと副プログラム(サブルーチンと関数)は, 両方を一つのファイルに書くこともできるが, それぞれを別ファイルに書くこともできる.
- しかし, 長いファイルから目的とする編集点を探すのは手間になり, プログラム作成の効率を下げるかもしれない.

複数ファイルに分けたプログラム

summationsub_main.f90

```
program summationsub

  implicit none

  integer :: num

  ! 1 から 10 までの和の計算
  call calcsun(10,num)
  write( 6, * ) num

end program summationsub
```

summationsub_sub.f90

```
! サブルーチンの定義
subroutine calcsun(n,kazu)

  implicit none

  integer, intent(in) :: n
  integer, intent(out) :: kazu

  integer :: i

  kazu = 0           ! kazu の初期化
  do i = 1, n
    kazu = kazu + i
  end do

end subroutine calcsun
```

複数ファイルに分けたプログラム

- 複数のファイルに分けたプログラムは, 下のようになるとコンパイルできる.

```
$ gfortran -o summationsub summationsub_main.f90 summationsub_sub.f90
```

- 用いるファイル名を並べる.
- 分割するファイルの数が増えると, 上のようにコンパイルするのは困難である. そんなときには, make コマンドを使うと良い.
 - make は多数のファイルで構成されたプログラムをコンパイルするためのプログラムである.
 - ここでは詳細は説明しない.

実習

- 実習を通して, サブルーチンや関数に慣れましょう.