

C 言語 と Fortran の構文比較

関数、サブルーチン

C 言語

関数の呼び出し

```
m = myfunc(n);  
i = mysub(&a, b, c);
```

&a は **a** のアドレス
b, c は値を表わす

関数の定義

```
型名 関数名 (型名 引数){  
処理  
}
```

```
int myfunc(int i){  
    return i+1;  
}
```

```
int mysub(double *x, double y, double z){  
    *x = y + z;  
    return 0;  
}
```

***x** はアドレス **x** の値を表す
y, z は局所変数で return 後に消滅

a のアドレスを渡すと **a** と **x** は同じの実体をもつ。
b, c のように値を渡すと実体は異なったものになる。
上の例では **b** と **y** は呼び出し時に値だけがコピーされ、
その後 **y** が変化したとしても **b** には反映されない。

Fortran

関数呼び出し

```
integer m, n  
m = myfunc(n)
```

引数、返却値の型に注意

関数の定義

```
型名 function 関数名 (引数)  
引数の定義  
処理  
return  
end
```

```
integer function myfunc( i )  
integer i  
myfunc = i+1  
return  
end
```

サブルーチンの呼び出し

```
real*8 a, b, c  
call mysub(a,b,c)
```

引数の型に注意

サブルーチンの定義

```
subroutine サブルーチン名 (引数)  
引数の定義  
処理  
return  
end
```

```
subroutine mysub(x, y, z)  
real*8 x, y, z  
x = y + z  
return  
end
```

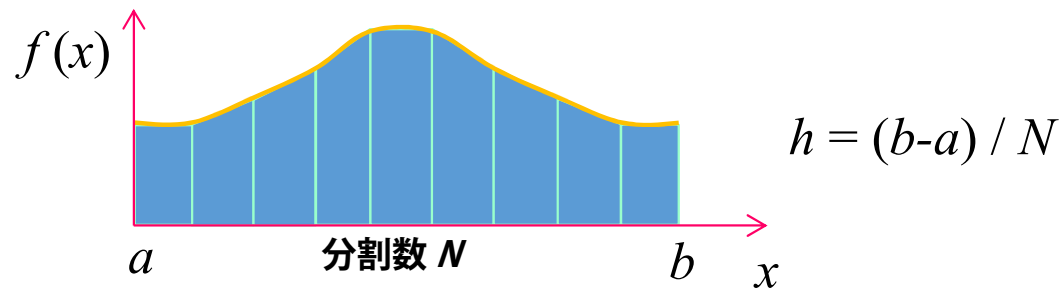
Fortran における引数の実体は、変数のアドレスで、
例えば引数が配列の場合、先頭の要素へのポインタになっている

積分計算 (関数を利用するプログラムの例)

$$I = \int_a^b f(x) dx$$

台形公式

$$I = h \left[\frac{1}{2} f(a) + \sum_{n=1}^{N-1} f(a + nh) + \frac{1}{2} f(b) \right]$$



分割数 N を多くしたときの結果の変化から収束判定を行うことで、最適な N を自動的に決めることができる。

```
real*8 :: s, h, f, PI
integer :: i, N=100
```

```
PI=4.d0*atan(1.d0)
```

```
h=PI/2/N
```

```
s=(f(0.d0)+f(PI/2))/2
```

```
do i=1, N-1
```

```
    s=s+f(i*h)
```

```
end do
```

```
write(*,*) s*h
```

```
end
```

$$I = \int_0^{\pi/2} \cos(x) dx$$

```
real*8 function f(x)
real*8 :: x
```

```
f =cos(x)
```

```
return
```

```
end
```

ただし、分割数 N を多くしたときの（情報落ち）誤差に注意

積分区間が無限の場合には、積分範囲を広げながら収束を判定するか、変数変換によって有限区間の積分に直して実行する。

C言語 と Fortran の構文比較

配列、ポインタ

C言語

5個の要素をもつ整数型配列 a の宣言

例 1 `int a[5];`

例 2 `int a[5]={1,1,1,1,1};`

例 3 `int a[ ]={1,1,1,1,1};`

個々の要素は

`a[0]` 配列の先頭の要素 `a[0]`
`a[1]` の値が保存されている
`a[2]` メモリ上のアドレスは
`a[3]` `a` で参照できる
`a[4]`

メモリ上の配置

`a[0][0]`
`a[0][1]`
`a[1][0]`
`a[1][1]`
`a[2][0]`
`a[2][1]`
`a[3][0]`
`a[3][1]`
`a[4][0]`
`a[4][1]`

配列の名前 `a` は
先頭アドレスを表す ポインタ

多次元配列の宣言

`int a[5][2];`

Fortran

5個の要素をもつ整数型配列 a の宣言

例 1 `integer a(5)`

例 2 `integer :: a(5)=(/1,1,1,1,1/)` Fortran 90

例 3 `integer :: a(5)=1` すべての要素を1で初期化

個々の要素は

`a(1)` 配列の先頭の要素 `a(1)`
`a(2)` の値が保存されている
`a(3)` メモリ上のアドレスは
`a(4)` `a` で参照できる
`a(5)`

メモリ上の配置

`a(1,1)`
`a(2,1)`
`a(3,1)`
`a(4,1)`
`a(5,1)`
`a(1,2)`
`a(2,2)`
`a(3,2)`
`a(4,2)`
`a(5,2)`

配列の名前 `a` は
先頭アドレスを表す ポインタ

多次元配列の宣言

`integer a(5,2)`

C 言語 と Fortran の構文比較

変数のスコープ

C言語

初期のC言語では、変数の宣言は最初の実行文の前までに行うが、C99やC++では、どこで行ってもよい。

ブロック内で宣言された変数の有効範囲（スコープ）は宣言した時点からブロックの最後まで

関数の外側で宣言された変数のスコープは宣言した場所からソースファイルの最後まで

ブロックの内外で変数名が重複した場合は、内側が優先され、外側の変数は内側のブロックが終わるまで見えなくなる

Fortran

変数の宣言は最初の実行文の前までに行う

関数やサブルーチンの中で新しく宣言された変数のスコープはその関数やサブルーチンの中に限定される

複数の関数やサブルーチンの中で変数を共有するためには引数として引き渡すか、common 文を用いる
Fortran90以降ではグローバル変数モジュールを利用することも可能

C++言語 main.cpp

```
#include <cstdio>
int func(void);
using namespace std;
int i=100;

int main(void){
    int i=0, j=0;
    for (int i=1; i<=10; i++){
        j+=i;
    }
    printf("%5d %5d ¥n", i, j);
    j=func( );
    return 0;
}

int func(void){
    printf("%5d¥n", i);
    return 0;
}
```

変数のスコープの範囲
ファイルスコープ
ブロックスコープ