



情報処理演習II

No. 9

2023. 10. 24

芝浦工業大学 システム理工学部 機械制御システム学科

担当：桑原

「関数」に関する補足

•関数（サブルーチン）の呼出し

- ・サブルーチンから別のサブルーチンを呼び出すことも可能。
ただし、サブルーチンからメインルーチンを呼出すことはNG。
また、実行は呼出し元をたどって順次戻される。
- ・関数内で別の関数は定義できない。

• 引数、戻り値の型

- ・「呼出し側で渡す引数のデータ型」＝「関数の引数のデータ型」
- ・「関数の戻り値のデータ型」＝「戻り値を受ける変数のデータ型」
- ・戻り値は必ず 1 つ。

• 引数（データ）は「変数」でなく「値」で渡される

引数（変数）をそのまま渡しているのではなく、変数の**値**をコピーして渡している点に注意。

このことを **Call by value** と言う。

（今回からの内容では、データのもう一つの渡し方を学ぶ。同時に、引数や戻り値に配列変数を指定する方法も身に付ける）

例題

例：2つの変数に格納されている値を入替える関数を作りなさい。

サンプルプログラム 9.1

```
#include <stdio.h>
void swap(int x, int y){
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}

int main(void){
    int x = 1, y = 2;
    printf("Before: ¥n");
    printf("x = %d, y = %d¥n", x, y);

    swap(x, y);

    printf("After: ¥n");
    printf("x = %d, y = %d¥n", x, y);
    return 0;
}
```

(実行結果)

Before:
x = 1, y = 2
After:
x = 1, y = 2

値は入替わっていない? なぜ???

例題 うまいかなかった理由（１）

main関数内の変数 x、y と swap関数内の変数 x、y は別物

（それぞれ独自に宣言している！）

```
swap(int x, int y)
{
    中略
}

int main(void)
{
    int x = 1, y = 2;

    ～中略～
    swap(x, y);
    ～中略～

    return 0;
}
```

別物
コンパイルは通る。

例題 うまいかなかった理由（２）

- 関数の引数は**値渡し**になる（**Call by value**）

```
#include <stdio.h>
void swap(int x, int y){
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}

int main(void){
    int x = 1, y = 2;
    printf("Before: ¥n");
    printf("x = %d, y = %d¥n", x, y);

    swap(x, y);

    printf("After: ¥n");
    printf("x = %d, y = %d¥n", x, y);
    return 0;
}
```

関数 swap の呼出し時は、変数 x, y 自体が関数 swap に渡されるのではなく、実際は変数 x, y の値がコピーされて渡される。つまり、実際には「swap(1, 2);」が実行される。

変数そのものでなく、値のコピーを渡している!!!

例題 うまいかなかった理由（3）

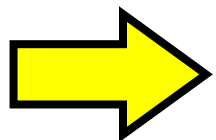
- main 内で変数 x , y を宣言し、それぞれに1と2を代入。
- printf で main 内での変数 x , y の値を表示。
- swap(1,2)の実行
 - 1が swap関数内の変数 x (\neq main の変数 x) に代入され、
2が swap関数内の変数 y (\neq main の変数 y) に代入される。
 - swap関数内の変数 x の値と 変数 y の値が入替えられる。
つまり swap関数内の変数 x の値は 2、変数 y の値は 1。
- printf で main内 の変数 x と main の変数 y の値が表示される。
つまりそれぞれ1と2。

結局、何をしていたのか？

- 関数を呼び出したが、関数で使われた変数ではなく、main内で定義した変数を表示。
- 当然、値を入替えることはできない。

解決策

- 関数 swap 内から main 内の**変数** x, y **自体**を指定して、その値を入替えれば良い



では、どうやって**変数自体**を指定すれば良いか？

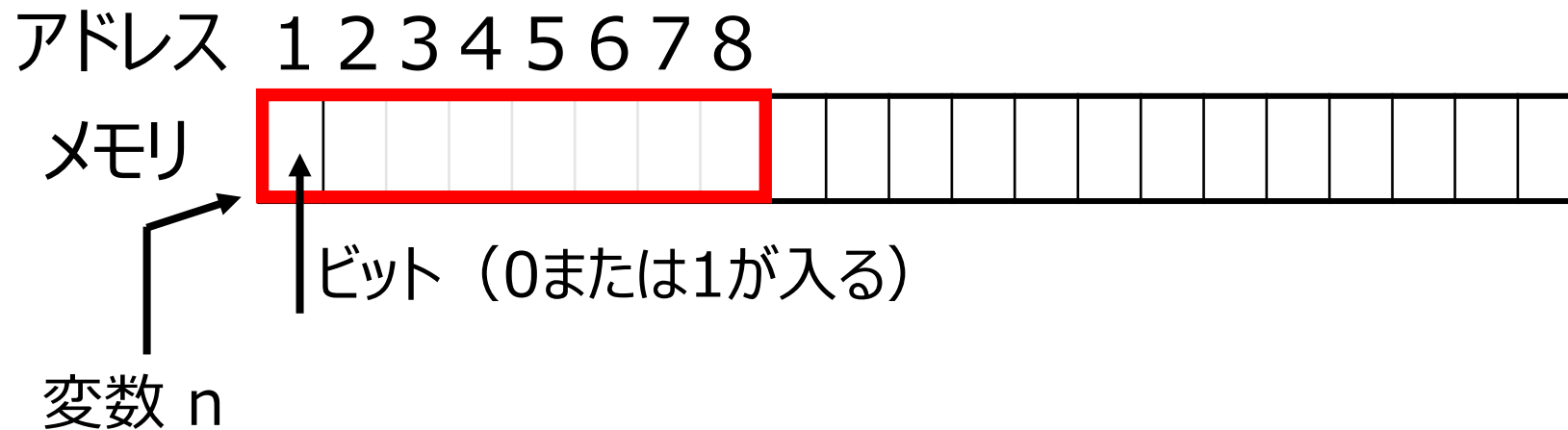
(変数名を指定しても、結局は中身の値をコピーして渡してしまう)

変数自体を指定するために、変数についてもう少し調べてみよう。

メモリ上の変数

- **変数**は、物理的には**メモリ上の領域**である。
- メモリ上にはアドレス（住所）があり、それを指定することでメモリ上の領域を指定できる。
- よって**メモリ上のアドレス**を指定すれば、**変数自体を指定できる**。

（実際のアドレスは16進数表示）



- この場合、変数 n はアドレス 0 番から7番までのビットからなる領域を表す。
- 変数が使うビット長さは変数の型によって決まる。

変数のポインタ

各変数が使用するビット数は、変数型から自動的に決まる。

よって・・・

変数を指定するには、
その変数が使用する **先頭のビットのアドレス** を指定すればよい。

=「（変数の）ポインタ」

この「変数が使用する先頭のビットのアドレス」を、
その変数の**ポインタ**と呼ぶ。

アドレス演算子

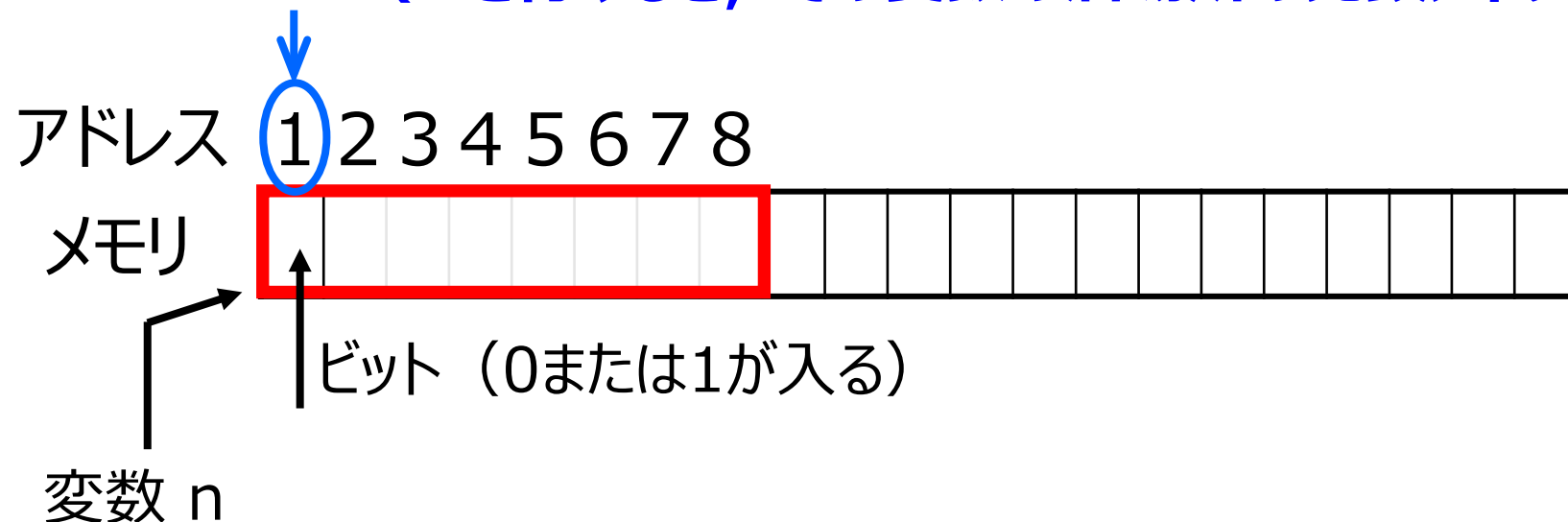
アドレス演算子（記号は&）

= 変数のポインタを求める演算子。変数名の**前**に置く。

例： 変数nに対して「**&n**」は**変数 n のポインタ**

（つまり変数nが使用する先頭のビットアドレス）を表す。

&n（&を付けると、その変数の居場所の先頭アドレス）



サンプルプログラム 9.2 各自で作成・実行してください。

サンプルプログラム 9.2

```
#include <stdio.h>

void main(){
    int  x = 1, size;

    printf("変数 x の値は %d です. ¥n", x);
    printf("変数 x のポインタは %p です. ¥n", &x);

    size = sizeof (x) * 8; /* 変数xのメモリ上の領域をビットで計算 */
    printf("変数 x が使用するビットの数は %d です. ¥n", size);
}
```

- 変数のポインタを printf で表示するときは、「%p」で指定する。
- sizeof(変数名) = 「その変数を使用するバイト数」を int 型の値で返す。
(これを8倍すれば、その変数を使用している領域の長さがビット単位で求められる。)

scanf 関数（復習）

例) scanf("%lf", &x); ← 変数 x のポインタを指定！

- 変数 x の値（倍精度浮動小数点数）をキーボード入力から読み取る。
- printf の文法に似ているが、変数 x の前に「&」がついて「&x」となる。

サンプルプログラム

```
(前略)
int main(void)
{
    double x;

    printf("Please input: x =¥n");
    scanf("%lf", &x);
    printf("x = %f.¥n", x);

    return 0;
}
```

注意：「&」が抜けてもコンパイルエラーにはならないが、実行しても予定通りの動作をしない

注意：変換仕様はprintf と若干異なる！

変換仕様	データ型
%s	文字列（末尾に¥0追加）
%d	整数型
%f	単精度浮動小数点数
%lf	倍精度浮動小数点数

ここまでのまとめ

サンプルプログラム 9.1

```
#include <stdio.h>
void swap(int x, int y){
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
}

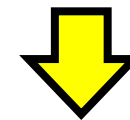
int main(void){
    int x = 1, y = 2;
    printf("Before: %n");
    printf("x = %d, y = %d%n", x, y);

    swap(x, y);
    printf("After: %n");
    printf("x = %d, y = %d%n", x, y);
    return 0;
}
```

関数の引数は値渡し

(Call by value)

- 関数に、値ではなく、変数自体を渡す方法：
変数のポインタを指定すれば良い
- 関数に変数のポインタを渡す
(つまり変数のポインタを引数として与える) には・・・？
→ **変数のポインタを格納する変数が必要！**
それはどんなデータ型？ 宣言や代入の方法は？



ポインタ変数 (単にポインタとも呼ぶ)

ポインタ変数の宣言（１）

- ポインタ = メモリ上のアドレス（住所）
- ポインタが指定する変数の型によって、ポインタの種類も異なる
 - int 型の変数には「int 型変数のポインタ」
 - float 型の変数には「float 型変数のポインタ」
 - double 型の変数には「double 型変数のポインタ」
 - char 型の変数には「char 型変数のポインタ」
- ポインタの種類によって、それらを格納するポインタ変数の種類も異なる
 - int 型変数のポインタには「int 型変数のポインタを格納するポインタ変数（略して int 型へのポインタ）」
 - float 型変数のポインタには「float 型変数のポインタを格納するポインタ変数（略して float 型へのポインタ）」
 - double 型変数のポインタには「double 型変数のポインタを格納するポインタ変数（略して double 型へのポインタ）」
 - char 型変数のポインタには「char 型変数のポインタを格納するポインタ変数（略して char 型へのポインタ）」

ポインタ変数の宣言（２）


重要

ポインタ変数の宣言（「*」を忘れずに！）

（変数の型宣言子） *（ポインタ変数名）；

→ ポインタ変数の格納するアドレス（ポインタ）には、宣言した型のデータが存在する

- 例：「int 型へのポインタ変数 p」の宣言例： `int *p；`

「宣言するポインタ変数が格納するポインタは、 `int *p；`
どの型の変数のアドレスか？」

ポインタ変数名 = 「p」

- 例：「double 型へのポインタ変数 p」の宣言例：
`double *p；`

重要

ポインタ変数への値の代入は

(ポインタ変数名) = (ポインタ) ;

- (復習) 「=」は「右を左に代入する」という意味を持つ。
数学の等号のように「等しい」という意味はない
- 右辺の「ポインタ」の先にある変数のデータ型と「ポインタ変数の種類 (型)」を一致させる

例) int n;

int *pn; (int 型へのポインタ変数 pn を宣言)

pn = &n; (pn に int 型変数 n のポインタ &n を代入する)

- 変数宣言と値の代入を1行で書くこともできる

例) int *pn = &n;

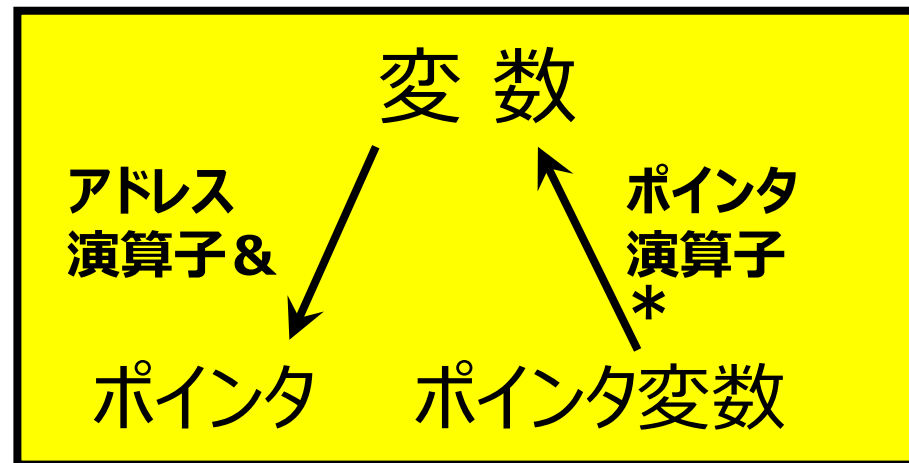
重要

ポインタ演算子（記号は *****）

→ ポインタ変数が指定する変数を求めることができる。

ただし、ポインタ変数の**前**に置く（アドレス演算子と同じ）。

例：ポインタ変数の名前が **p** のとき：
「***p**」 = ポインタ変数 **p** が指定する変数



ポインタ変数の必要性（確認）

- ある関数から別の関数の変数を操作（値の代入・参照等）をしたい場合には、ポインタを引数として渡すこと（Call by reference）が必要!!!
→ **Call by reference（ポインタ渡し）** を使わずに
他の関数内で使っている変数の値を参照・操作 することは不可能
（例題 サンプルプログラム9.1での失敗例を思い出そう）

サンプルプログラム 各自で書いて実行してみてください。

サンプルプログラム 9.3

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

```
void swap(int *x, int *y)
{
    int tmp;
    tmp = *x;
    *x = *y;
    *y = tmp;
}
```

引数はアドレスなので、**ポインタ変数で受ける**

「*x」はポインタxの先にある変数を指すが、メモリ上ではmain関数での変数x そのもの（「*y」でも同様）

```
int main(void)
{
    int x = 1, y = 2;

    printf("x = %d, y = %d\n", x, y);
    swap(&x, &y);
    printf("x = %d, y = %d\n", x, y);

    return 0;
}
```

関数swapには変数xとyの**ポインタを渡す**

戻り値以外で関数から値を複数取り出せる