



情報処理演習II

No. 10

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

前回までのまとめ

- ポインタ変数も、他の変数と同様に関数の引数として使用できる

a) 一般の変数が引数： 関数には変数の"値"が渡される

(Call by value)

→ 呼出し側の変数には一切影響を与えられない

b) ポインタ変数が引数： 関数には"ポインタ"（変数の先頭アドレス）が渡される

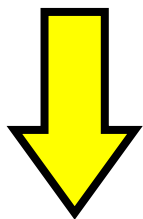
(Call by reference)

→ 呼出し側の変数に変化を与えることができる

- ある関数の中の変数の値を別の関数の中から変えるには、ポインタを引数として受渡す

プリプロセッサの利用

- プログラムの仕様変更の際に大変なので、配列の全要素数を効率よく変更する方法はないか？



- 「配列の大きさは、宣言時に定数として宣言しないといけない」が原則
- 定数ならOK → 後で変えられるようにして、コンパイル直前で定数に置換しては？
- **#define**命令：
プログラム中の（特定の文字（列）の）置換処理

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 10.1

```
#include <stdio.h>

#define NUMBER 5 /* Nの定義 */

int main(void)
{
    int data[NUMBER];
    int k=0;

    int arrayNumber = sizeof array / sizeof array[0];

    printf("arrayNumber: %d¥n", arrayNumber);
    return 0;
}
```

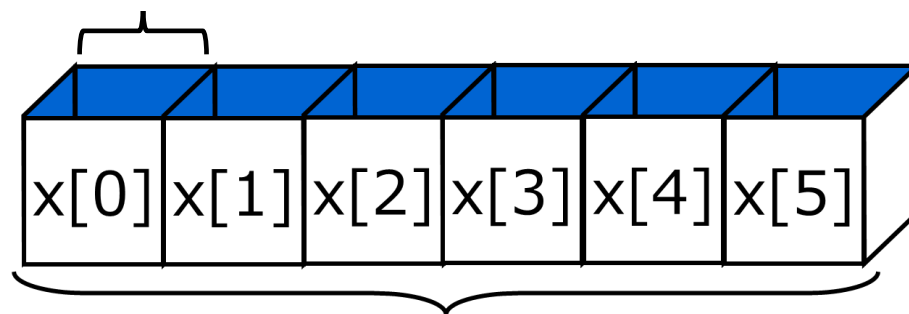
配列の個数を文字列定数「NUMBER」で定義し、
コンパイル直前に「5」に置換するように処理
→ プリプロセッサ #defineの働き
(プリプロセッサで扱う文字列は全て大文字で書くと"変数"と区別しやすい)

TABで空白を入れて見やすく (1つでなくても良い)

配列のデータサイズ

int型 のデータサイズ

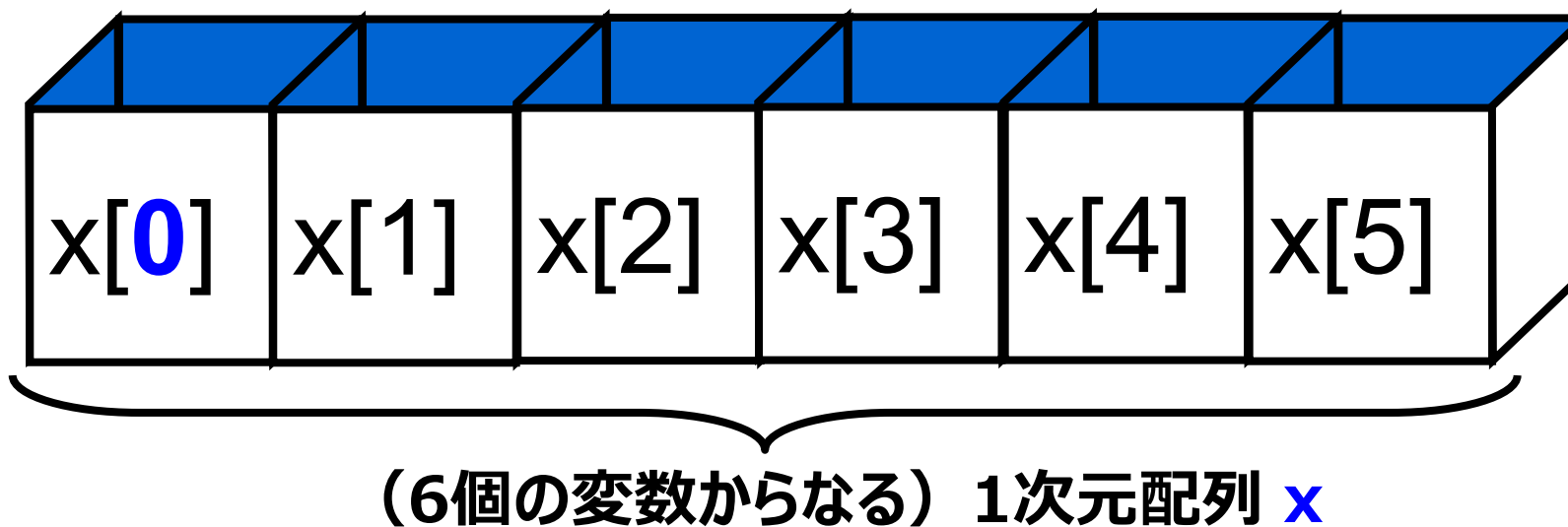
sizeof x[0]



sizeof x

配列の復習（１）

- **配列** = 同じ性質を持つ要素が一定の規則によって並んだ データの集合
- **複数の変数（箱）を列や行に並べたもの**
- 変数と同様に、**配列に名前をつける（配列の宣言）**
 - 名前の付け方のルールは変数と同様
- 変数を一列に並べたものを**1次元配列**と呼ぶ



配列の復習（２）

これが増えたことが
これまでの
"変数宣言"との違い

1次元配列の宣言：

（型宣言子） （配列名） [（要素数）]；

- 例： `int array[5];`
 - **5**つの **int 型**変数を要素に持つ配列変数 **array** ^(アレイ) を宣言
 - 変数の要素の名前はそれぞれ：
`array[0], array[1], array[2], array[3], array[4]`
（括弧内の番号は 0から始まることに注意！）
 - 配列の各要素となる変数を「**配列要素**」，各配列要素の括弧内の番号を「**配列の添え字**」と呼ぶ
- **配列宣言時の要素数に変数を用いることはできない**
 - ただし、配列の添え字には変数を用いてよい

配列の復習（3）

- 「各配列要素への値の代入」は、変数の場合と同様

例： `int array[5];` **1次元配列の宣言**

`array[0] = 1;`
`array[1] = 2;` } **1つずつ要素（変数）に値を代入**

- まとめて代入する方法もある（注意）

例： `int array[5] = { 1, 2, 3, 4, 5 };`

（これは `array[0] = 1, array[1] = 2, array[2] = 3, array[3] = 4, array[4] = 5` という意味）

- 規則的に値を代入する場合は、繰返し制御文（for文, while文など）を使っても良い

サンプルプログラム（再掲）

サンプルプログラム 8.1

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

```
void main()
```

```
{
```

```
    int i;
```

```
    int array[5];
```

```
    array[0] = 10;
```

```
    array[1] = 20;
```

```
    array[2] = 30;
```

```
    array[3] = 40;
```

```
    array[4] = 50;
```

```
    for (i = 0; i < 5; i++) {
```

```
        printf("%d 番目の要素変数の値は %d です\n", i+1, array[i]);
```

```
    }
```

```
}
```

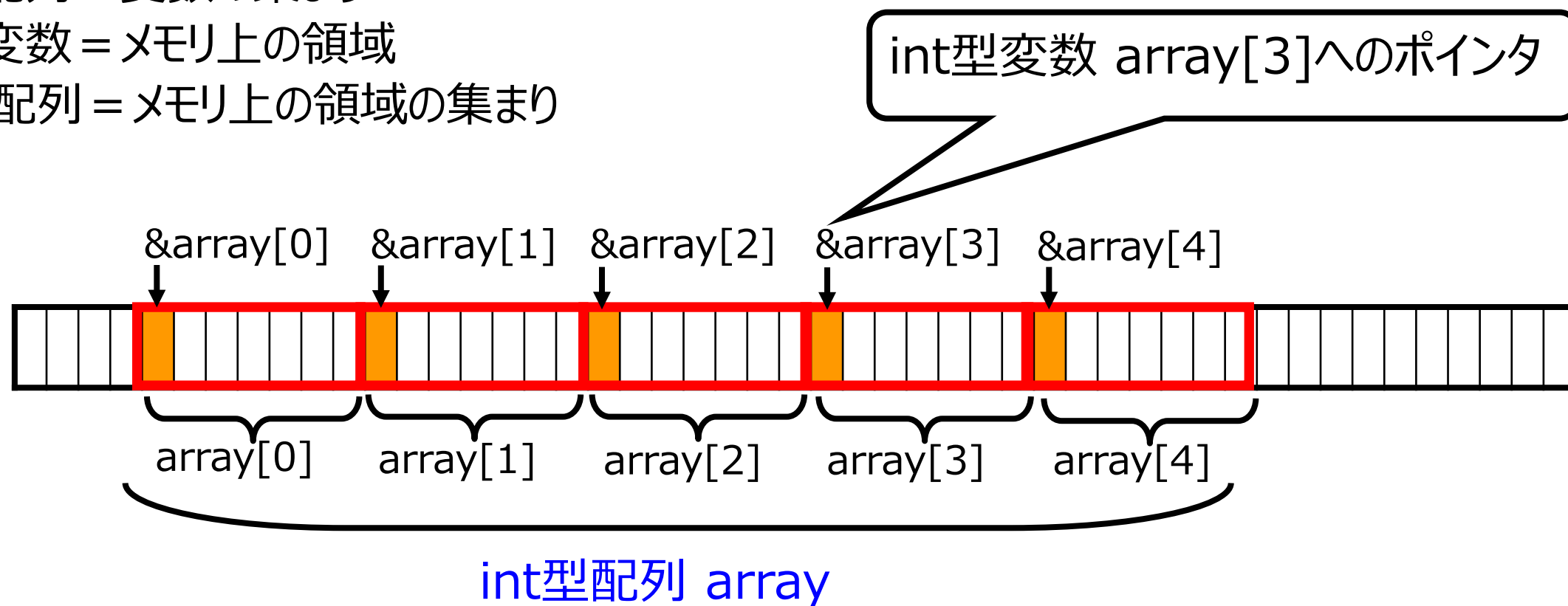
```
int  n=100;  
int  array[n];
```

は誤り!!!

（配列宣言時の個数には**定数**しか使えない）

配列とポインタ（１）

- 変数が使用する先頭のビットアドレスを、その変数の"ポインタ"と呼んだ
- 配列の場合は？
 - 配列 = 変数の集まり
 - 変数 = メモリ上の領域→配列 = メモリ上の領域の集まり

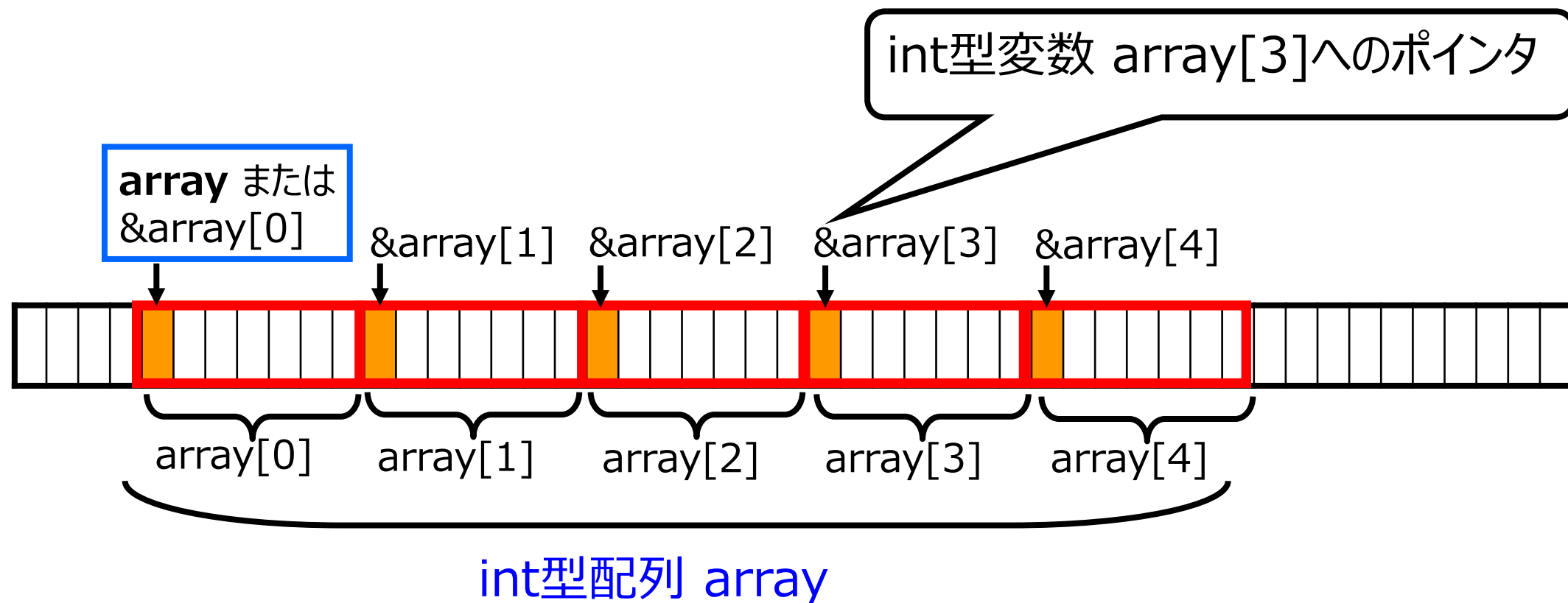


配列とポインタ（2）

- **配列名**は「その配列の**0番目の配列要素のポインタ**」を表す

（これまでは、配列名自体をプログラム中で扱うことは無かったが、実はこのような意味がある）

例： 配列名 `array` は `&array[0]` と同じ意味を持つ！



サンプルプログラム

サンプルプログラム 10.2

```
#include <stdio.h>

int main(void)
{
    int i, array[5];

    for (i = 0; i < 5; i++) {
        printf("配列要素 array[%d] のポインタは %p です. ¥n", i, &array[i]);
    }
    printf("配列名 array はポインタ %p を表します. ¥n", array);
    return 0;
}
```

実行結果の例

配列要素 array[0] のポインタは 0012FECC です.
配列要素 array[1] のポインタは 0012FED0 です.
配列要素 array[2] のポインタは 0012FED4 です.
配列要素 array[3] のポインタは 0012FED8 です.
配列要素 array[4] のポインタは 0012FEDC です.
配列名 array はポインタ 0012FECC を表します.


array[0]とarrayの
ポインタは一致
している

ポインタ変数の足し算（１）

- ポインタ変数の演算の意味すること

例： `int n = 1;`

`int *p=&n;`

`p++;`  ポインタ変数 p をインクリメントすると？

ポインタ 変数 p に対して

`p++ (p+1)` : ポインタ変数 p が指定する変数 (領域)の1つ次の変数 (領域)

`p-- (p-1)` : ポインタ変数 p が指定する変数(領域)の1つ**前**の変数 (領域)

ポインタ変数の足し算（引き算）は、配列を操作するときに便利！

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 10.3

```
#include <stdio.h>

int main(void)
{
    int i;
    int array[5] = {4, 2, 3, 1, 7};

    for (i = 0; i < 5; i++) {
        printf("array[%d] = %d¥n", i, *(array + i));
    }

    return 0;
}
```

ポインタ変数の足し算（２）

```
int  
array[5];  
int *p;  
p = array;
```

としたとき、

- p と $array$ と $\&array[0]$ は全て等価
- $p + 1$ と $\&array[1]$ は等価

また、 int 型変数 i に対しても、

- $p + i$ と $\&array[i]$ は等価
- $*(p + i)$ と $array[i]$ は等価

さらに、 $*(p + i)$ は $p[i]$ と書くこともできる（これは知っておくと便利）。

配列を関数の引数に使用方法

- 配列を渡される関数の引数はポインタ変数で定義する。

配列を関数に渡す側

```
#include <stdio.h>
#define N 5

int main(void) {
    int i, total;
    int array[N];
    printf("%d 個の整数を入力して下さい。 ¥n", N);
    for (i = 0; i < N; i++) {
        scanf("%d", &array[i]);
    }

    total = sum(array); 配列渡し

    printf("合計は %d です。 ¥n", total);

    return 0;
}
```

配列を渡される側 **ポインタ変数**

```
int sum(int *array) {
    int i, sum = 0;

    for (i = 0; i < N; i++) {
        sum = sum + array[i];
    }

    return(sum);
}
```

**渡された配列を参照、
値の代入が可能**

注意

- 配列名はポインタ（アドレス）ではあるが、ポインタ変数（変数）ではない。
→ 変数のように値を代入したりすることはできない。

- 例 :

```
int aa[3] = { 1 , 3 , 5 };  
int ab[3];  
ab = aa;
```

 ← 配列名はポインタ**変数**では**ない**ので、
配列名 ab には値を代入できない。

- 例 :

```
int aa[3] = { 1 , 3 , 5 };  
int *p; // ポインタ変数  
p = aa;
```

 ← p はポインタ変数なので、これはOK。

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 10.4

```
#include <stdio.h>
#define N 5          /* defineにより、コンパイル前に文字列Nを5に置換 */

int sum(int *array)   /* 整数型関数sumは、配列変数を引数として総和を返す */
{
    int i, sum = 0;

    for (i = 0; i < N; i++) {
        sum = sum + array[i];
    }      /* ↑ただし、arrayはあくまでポインタ変数である点に注意 */
    return(sum);
}

int main(void)
{
    int i, total;
    int array[N];

    printf("%d 個の整数を入力して下さい。 ¥n", N);
    for (i = 0; i < N; i++) {
        scanf("%d", &array[i]);
    }

    total = sum(array);
    printf("合計は %d です。 ¥n", total);

    return 0;
}
```