

Lecture 8

プログラミング演習Ⅰ その8

本日の演習の流れ

配列の利用

course page

<http://amth.mind.meiji.ac.jp/courses/PE1/>

演習問題の目安時間について

1. 10分
2. 10分
3. 10分
4. 20分
5. 10分
6. 20分
7. 20分
8. 20分

配列 (arrays) について

EX

100人の学生の成績にそれぞれ変数（例えば, x_1, x_2, \dots, x_{100} ）を割り振ってしまうと、プログラムを入力する手間は大変なものになり、それだけ間違いも起こりやすくなる。

データ処理において、1つ1つのデータに変数名をつけて、メモリに記憶しておかないと処理できない場合がある。少量のデータときには、1つずつ変数名をつけることもできるが、データ量が増えると困難になる。

⇒

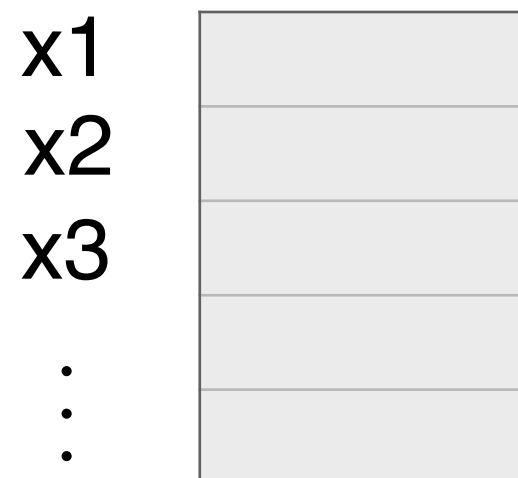
大量のデータを容易に処理する方法として配列 (arrays) について学ぶ。

配列 (arrays) について

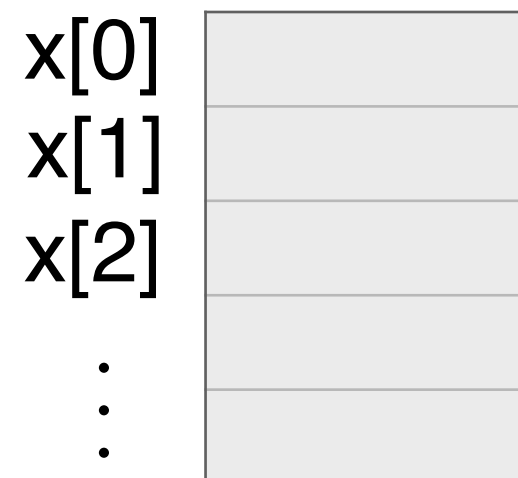
図aのように x_1, x_2, \dots と変数名をつける代わりに, 図bのように記憶場所のグループに x という名前をつける. この記憶場所のグループを配列といい, この名前を配列名にいう. 配列の個々の記憶場所を

$x[0], x[1], x[2], \dots$

と表し, これを配列要素という. 配列名の後のかっこ内の数字は添字 (index) といい, 配列の何番目の要素になるかを示す. 配列の添字は0から始まる.



(図a) 変数



(図b) 配列

配列 (arrays) について

配列には、変数と同様、整数型 (int) と実数型 (double) があり、使用する際にはプログラムの冒頭で宣言する必要がある。宣言によって、配列のためのメモリー領域が確保される。

配列の宣言の構文

型 配列名 [要素数];

例) int x[5];

注意：添字は0から始まり、最後は 要素数-1 の数で終わる。

NOTE：添字のことをインデックス (index) と呼ぶこともある。

配列の初期化について

配列のデータがあらかじめ分かっている場合には，配列宣言するときに，データを配列にセットすることができる．これを配列の初期化 (array initialization) という．

1 次のプログラムを実行してみよう．

```
1 #include <stdio.h>
2
3 int main(void){
4     int x[5] = {10, 13, 11, 25, 8};    // 配列の宣言と初期化
5     printf("%d\n",x[0]);
6     printf("%d\n",x[2]);
7     printf("%d\n",x[4]);
8     return 0;
9 }
```

x =	10	13	11	25	8
	x[0]	x[1]	x[2]	x[3]	x[4]

配列 (arrays) について

キーボードから配列の各要素にデータを入力するときには、for文を使うとよい。

2 次のプログラムを実行してみよう。

```
1 #include <stdio.h>
2
3 int main(void){
4     int i;
5     int x[5];
6     for(i=0;i<5;i++){
7         scanf("%d",&x[i]);
8     }
9     printf("values are: \n");
10    printf("%d\n",x[0]);
11    printf("%d\n",x[2]);
12    printf("%d\n",x[4]);
13    return 0;
14 }
```

このプログラムでは、次のようにscanf関数が5回実行される。

```
scanf("%d", &x[0]);
scanf("%d", &x[1]);
scanf("%d", &x[2]);
scanf("%d", &x[3]);
scanf("%d", &x[4]);
```


配列とデータの出力について

配列の各要素のデータの出力は次のようにfor文で行う.

3 次のプログラムを実行してみよう.

出力

```
1 #include <stdio.h>
2
3 int main(void){
4     int i;
5     int x[7] = {7,6,5,4,3,2,1};
6     printf("values are: \n");
7     for(i=0;i<7;i++){
8         printf("%d\n",x[i]);
9     }
10    return 0;
11 }
```

```
[ Lec8_Pro ./a.out
values are:
7
6
5
4
3
2
1
```

配列とデータの出力について

- 4 以下の配列の内容の合計と平均を計算するプログラムを作成せよ.

```
double arr[6] = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
```

定数式(constant expressions, macro)について

定数式とは、結果が定数になる式のところをいう。

```
#define N 3  
#define PI 3.14
```

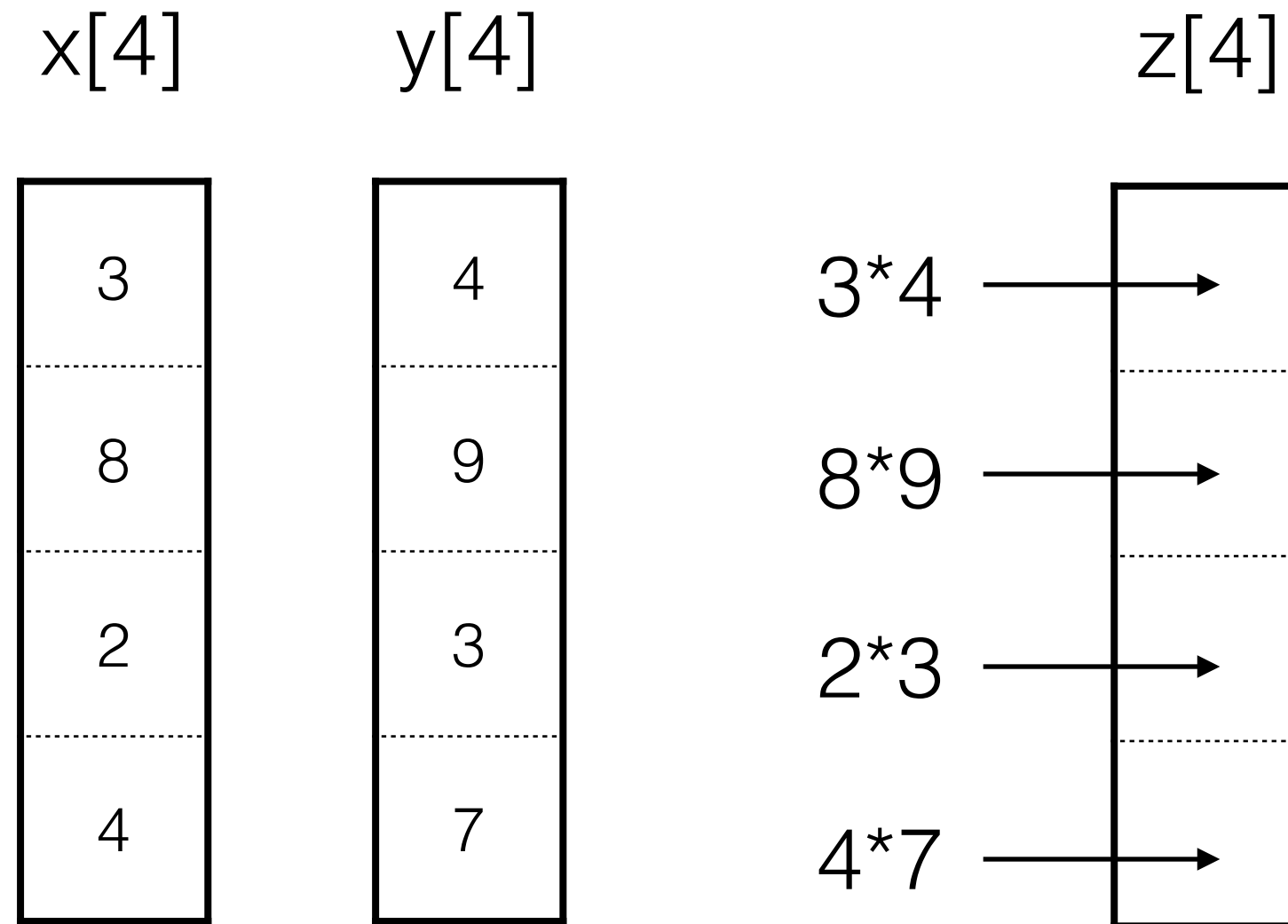
これらの定数式は、プリプロセッサによって、コンパイルに入る前に置き換えらる。式の終わりにセミコロン `;` をつけません。

5 次のプログラムを実行せよ。

```
1 #include <stdio.h>  
2 #define N 3  
3 #define PI 3.14  
4 int main(void){  
5     printf("The value of N is: %d\n",N);  
6     printf("The value of PI is: %lf\n",PI);  
7     return 0;  
8 }
```

配列とデータの出力について

- 6 以下のように配列 $x[4]$, $y[4]$ を作り, 各要素の積を要素とする配列 $z[4]$ をつくるプログラムを作成せよ.



配列とデータの出力について

- 7 2次元ベクトル a, b の成分をキーボードで入力し、ベクトル a, b の和と差を求めて表示するプログラムを作成せよ。

EX
$$\left. \begin{array}{l} a = (1.2, 7.4) \\ b = (8.6, 3.5) \end{array} \right\} \Rightarrow \begin{array}{l} a + b = (9.8, 10.9) \\ a - b = (-7.4, 3.9) \end{array}$$

ヒント 配列を使用する。

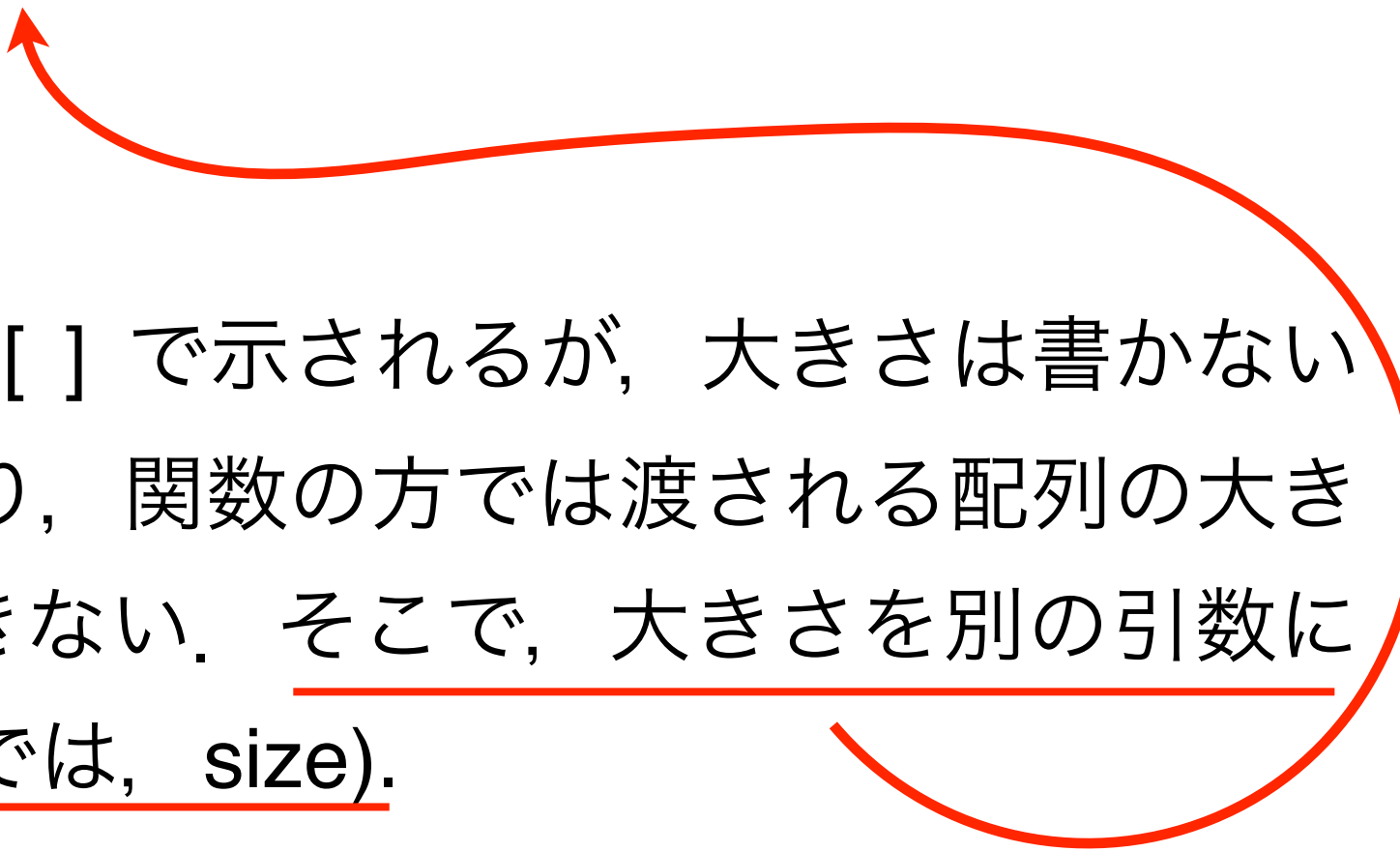
```
double a[2];
```

配列と関数について

配列の受け渡しについて見てみよう.

関数の先頭は次のようにする:

```
double myfunc(int x[], int size){  
    ...  
}
```



引数 x が配列であることは `[]` で示されるが、大きさは書かない (書いても無視される). つまり、関数の方では渡される配列の大きさを自動的に知ることができない. そこで、大きさを別の引数にするのが普通である. (ここでは、size).

関数を呼ぶ側は次のようにする:

```
double a[5] = {3.3, 5.1, 2.2, 6.8, 4.2};  
double biggest = myfunc(a, 5);
```

(配列名だけを書く)

配列と関数について

8 配列の最大値を求める関数を作成せよ.

ヒント

```
1 #include <stdio.h>
2
3 double maxval(double x[], int size){
4     ...
5 }
6
7 int main(void){
8     double a[5]={3.3, 5.1, 2.2, 6.8, 4.2};
9     double biggest = maxval(a,5);
10    printf("%lf\n",biggest);
11 }
```