生態模擬：以 C 語言為例

Class 08 (2018/05/17)

- Basics of Pointer
  - 8.1 Memory, Address, and Variable
  - 8.2 Pointer
  - 8.3 Parameter in function and Pointer
  - 8.4 Array and Pointer
  - 8.5 Homework for using Pointer
- Applications of Pointer and Array

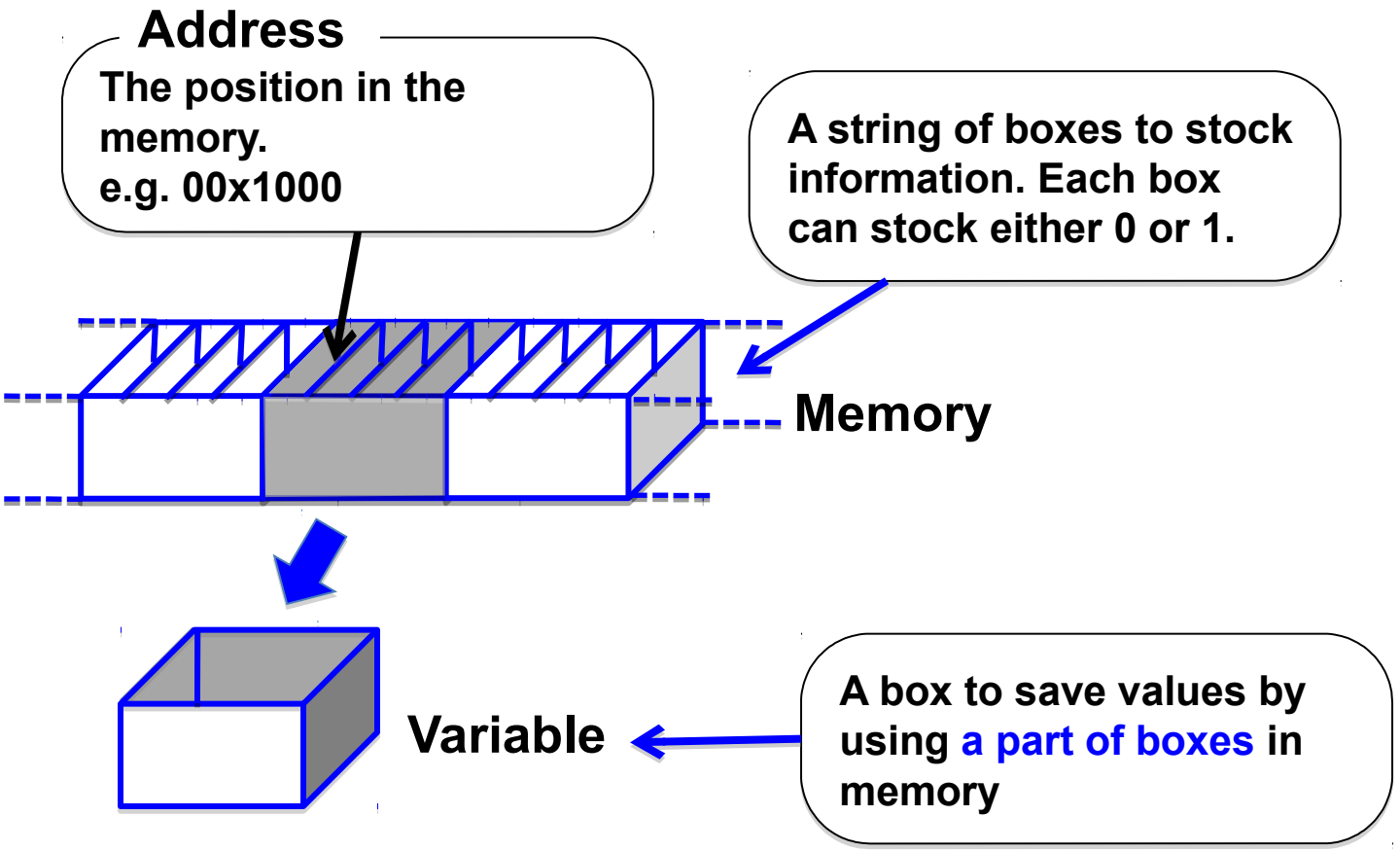
The next week….

Takeshi Miki

三木　健（海洋研究所）

8.1 Memory, Address, and Variable

The address in C represents the position in the memory, which is occupied by a variable.



8.1 Memory, Address, and Variable

Let's check the address of a variable.

```
#include <stdio.h>

int main(void)
{
        int a = 5;

        printf("The value of the variable a is %d.\n", a);
        printf("The address of the variable a is %p.\n", &a);

        return 0;
}
```

the unary address operator

8.2 Pointer

How to declare a *pointer variable* and how to *store* the address.

構文 (Syntax):

```
Datatype      *Pointer_name;
```

```
#include <stdio.h>

int main(void)
{
        int a = 5;
        int *pA;

        pA = &a;

        printf("The value of the variable a is %d.\n", a);
        printf("The address of the variable a is %p.\n", &a);
        printf("The value of pointer pA is %p.\n", pA);
        return 0;
}
```

Store the address of variable 'a' in pointer 'pA'

Now, pA points to a.

8.2 Pointer

We can know the value of variable, pointed by pointer variable.

構文 (Syntax):

```
*Pointer_name
```

```
#include <stdio.h>
int main(void)
{
        int a = 5;
        int *pA;
        pA = &a;

        printf("The value of the variable a is %d.\n", a);
        printf("The address of the variable a is %p.\n", &a);
        printf("The value of pointer pA is %p.\n", pA);
        printf("The value of *pA is %d.\n", *pA);
        return 0;
}
```

8.2 Pointer: a short summary

The concept of pointer is confusing…

a	Variable a
&a	Address of variable a

```
int *pA = &a;
```

(Declaration and initialization of pointer pA)

pA	Pointer, which stores the address of the variable a
*pA	Variable that is pointed to by the pointer storing the address of variable a

Variable a

*pA  a

Identical

一樣

8.2 Pointer

We can change the value of a variable, by using pointer

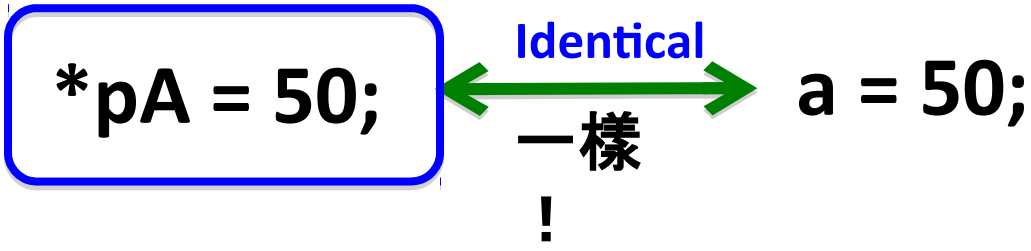
***pA** ⟷ **a** (Identical)

```
#include <stdio.h>
int main(void)
{
        int a = 5;
        int *pA = &a;

        *pA = 50;

        printf("The value of the variable a is %d.\n", a);
        return 0;
}
```

***pA = 50;** ⟷ **a = 50;** (Identical)

一樣

!



Why not directly changing the value of a???

→The pointer is necessary in functions!!

8.3 Parameter in function and Pointer

Prepare the following two functions and compare what happens.

```
void swap1 (int x, int y)
{
        int tmp;

        tmp = x;
        x = y;
        y = tmp;
}
```

functions cannot change values of variables

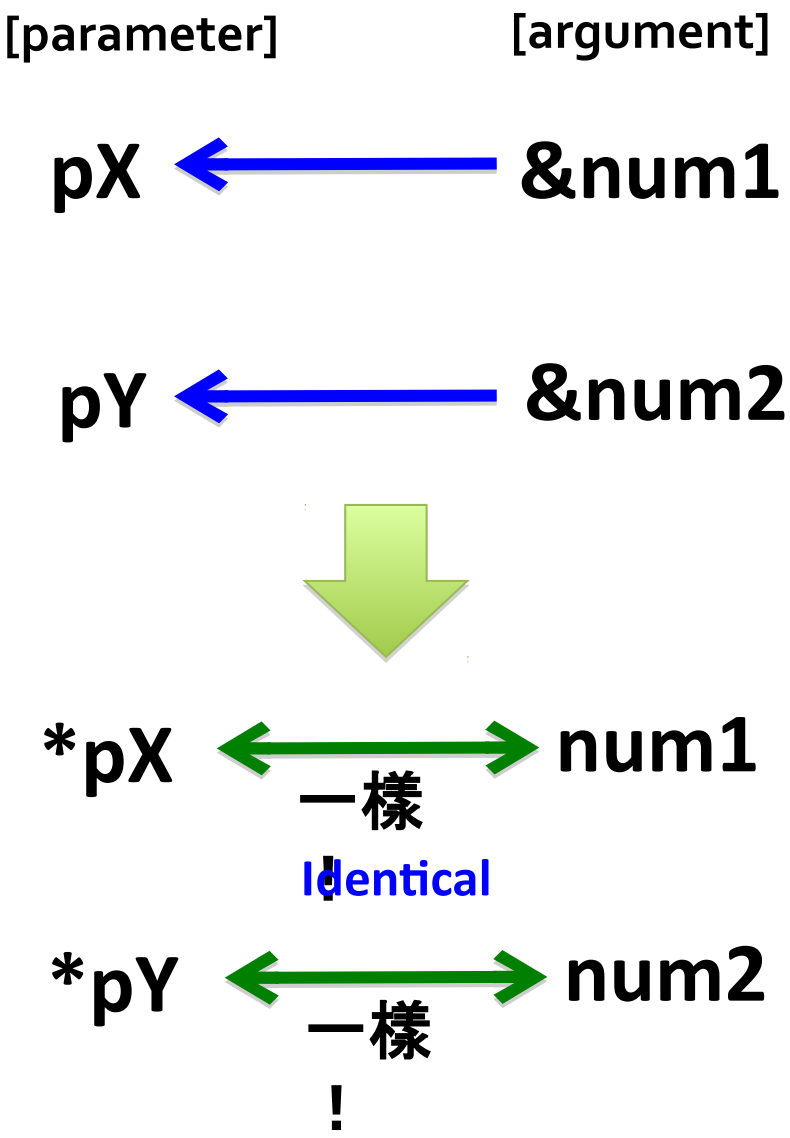
```
void swap2 (int *pX, int *pY)
{
        int tmp;

        tmp = *pX;
        *pX = *pY;
        *pY = tmp;
}
```

```
int main (void)
{
        int num1 = 5;
        int num2 = 10;
        .....
        //exchange the value of num1 and that of num2
        swap1 (num1, num2);
        swap2(&num2, &num2);
```

8.3 Parameter in function and Pointer

The relationship between parameter and argument

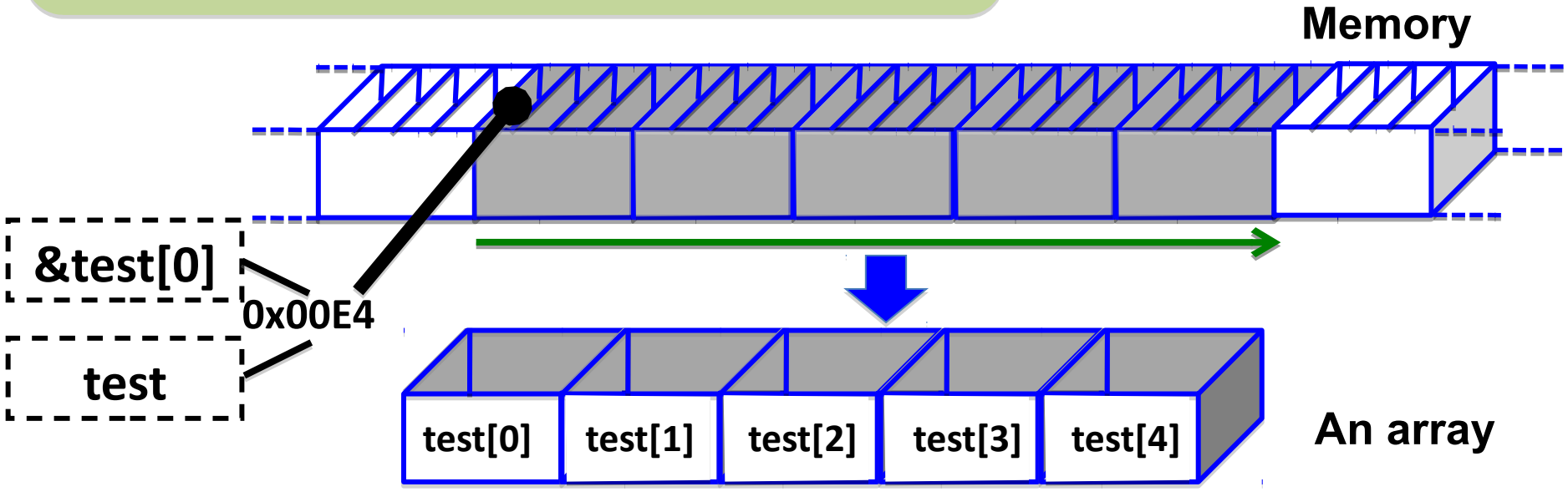
```
void swap2 (int *pX, int *pY)
{
        int tmp;

        tmp = *pX;
        *pY = *pX;
        *pX = tmp;
}
```



8.4 Array and Pointer

The relationship between the name of array and address

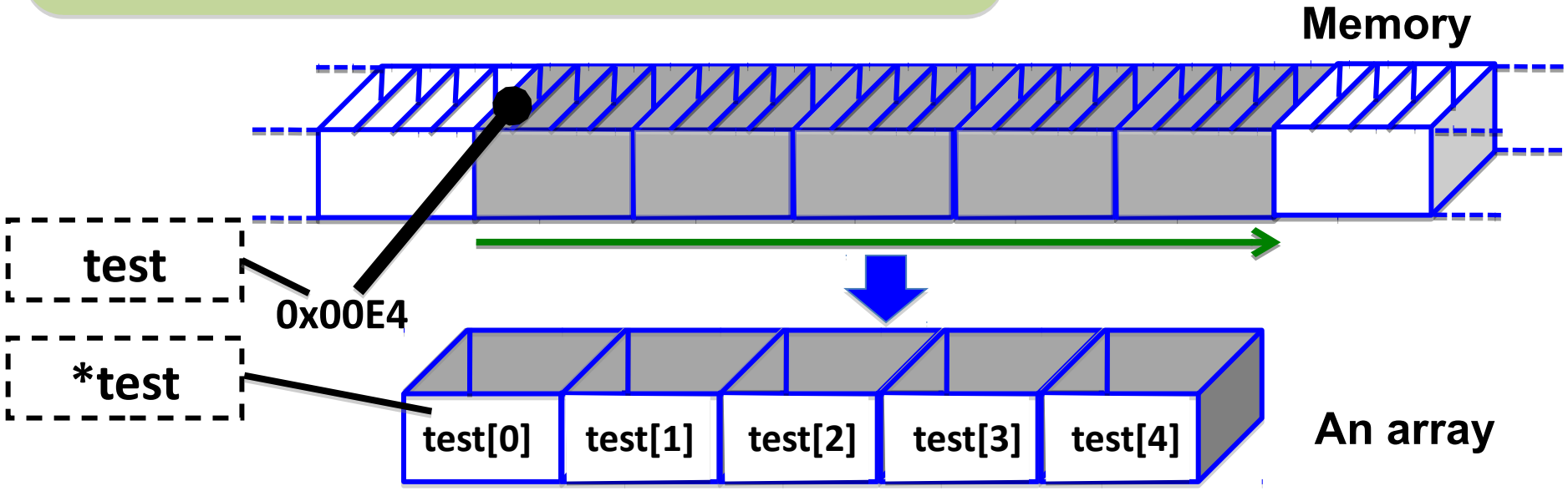
```
#include <stdio.h>
int main(void)
{
    int test[5] = {8, 6, 5, 2, 7};

    printf("The value of test[0] is %d.\n", test[0]);
    printf("The address of test[0] is %p. \n", &test[0]);
    printf("The value of test is %p. \n", test);
    return 0;
}
```



8.4 Array and Pointer

The relationship between the name of array and the value of the first element

```
#include <stdio.h>
int main(void)
{
    int test[5] = {8, 6, 5, 2, 7};

    printf("The value of test[0] is %d.\n", test[0]);
    printf("The address of test[0] is %p. \n", &test[0]);
    printf("The value of *test is %d. \n", *test);
    return 0;
}
```

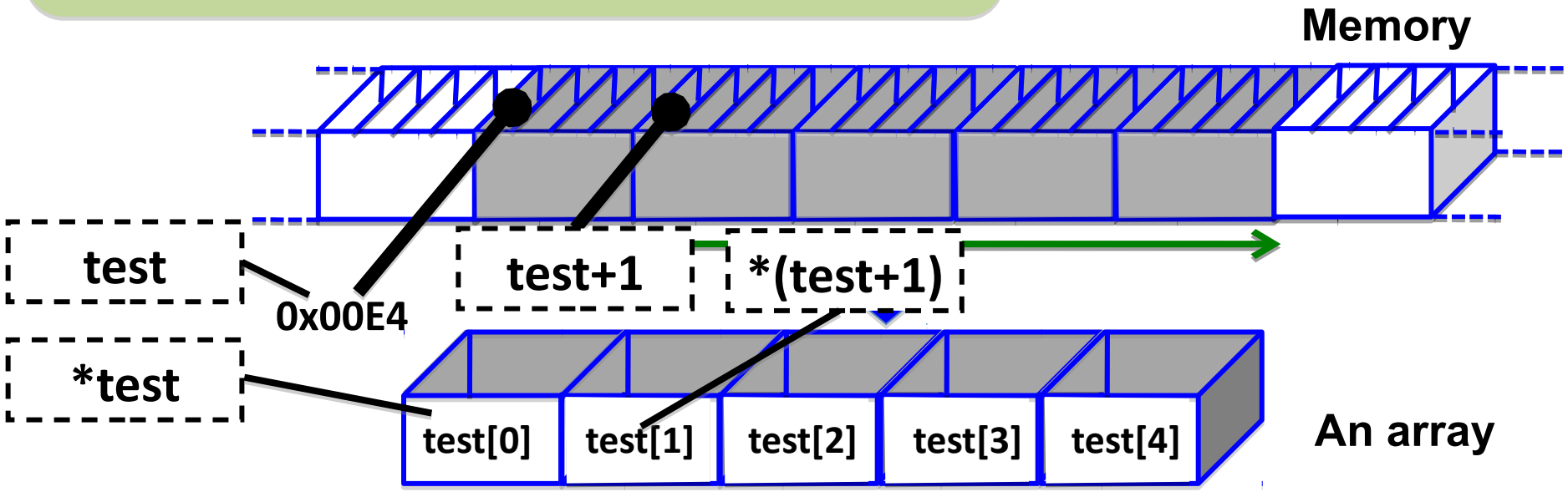


8.4 Array and Pointer

The pointer operators (+, -).

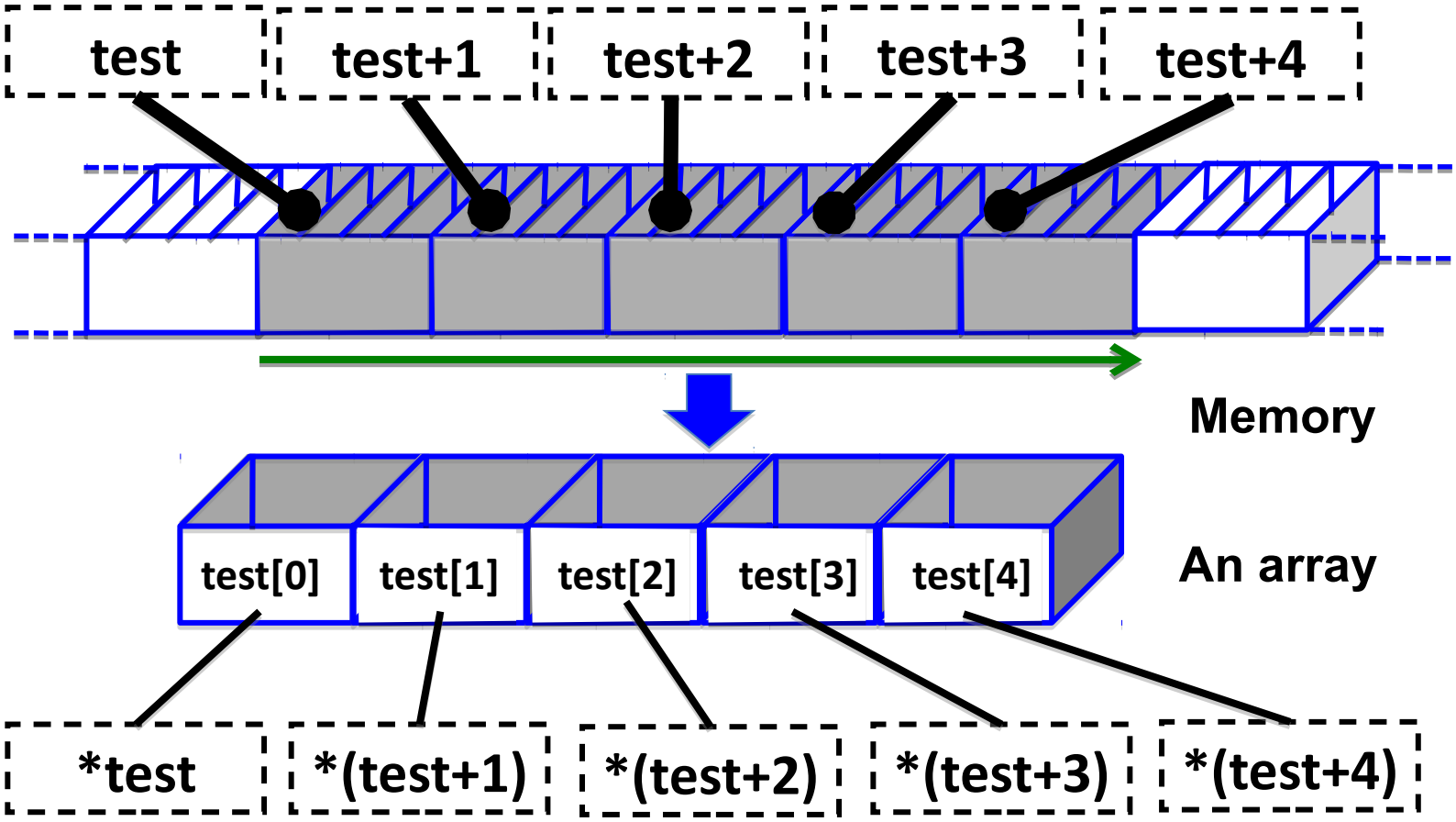
```
#include <stdio.h>
int main(void)
{
    int test[5] = {8, 6, 5, 2, 7};

    printf("The value of test  is %p.\n", test);
    printf("The test+1 is %p. \n", test+1);
    printf("The value of *(test+1) is %d. \n", *(test+1));
    return 0;
}
```

重要
(test+1) does NOT mean to add 1 to test !!



8.4 Array and Pointer: a short summary

The pointer operators (+, -).



8.4 Array and Pointer: dynamic allocation of memory to array (1)

The recommended way of the definition of 1-D array (i.e. vector)

```
#include <stdio.h>
#include <stdlib.h>       //need to be included
int main(void)
{
    double *x;          //pointer to double
    int size_v = 10;      //size of vector
    int j;
1   x = (double *) malloc( (size_t) ((size_v + 1)*sizeof(double)));
    if(x == NULL) {
        printf("The allocation was failed.\n");
        exit(1);
    } //error check is critical!!

2   for(j = 1; j <= size_v;j++) x[j] = 1.0*j;
3   free(x);  //must to release memory!!
    return 0;
}
```

To use functions related to memeory allocation.

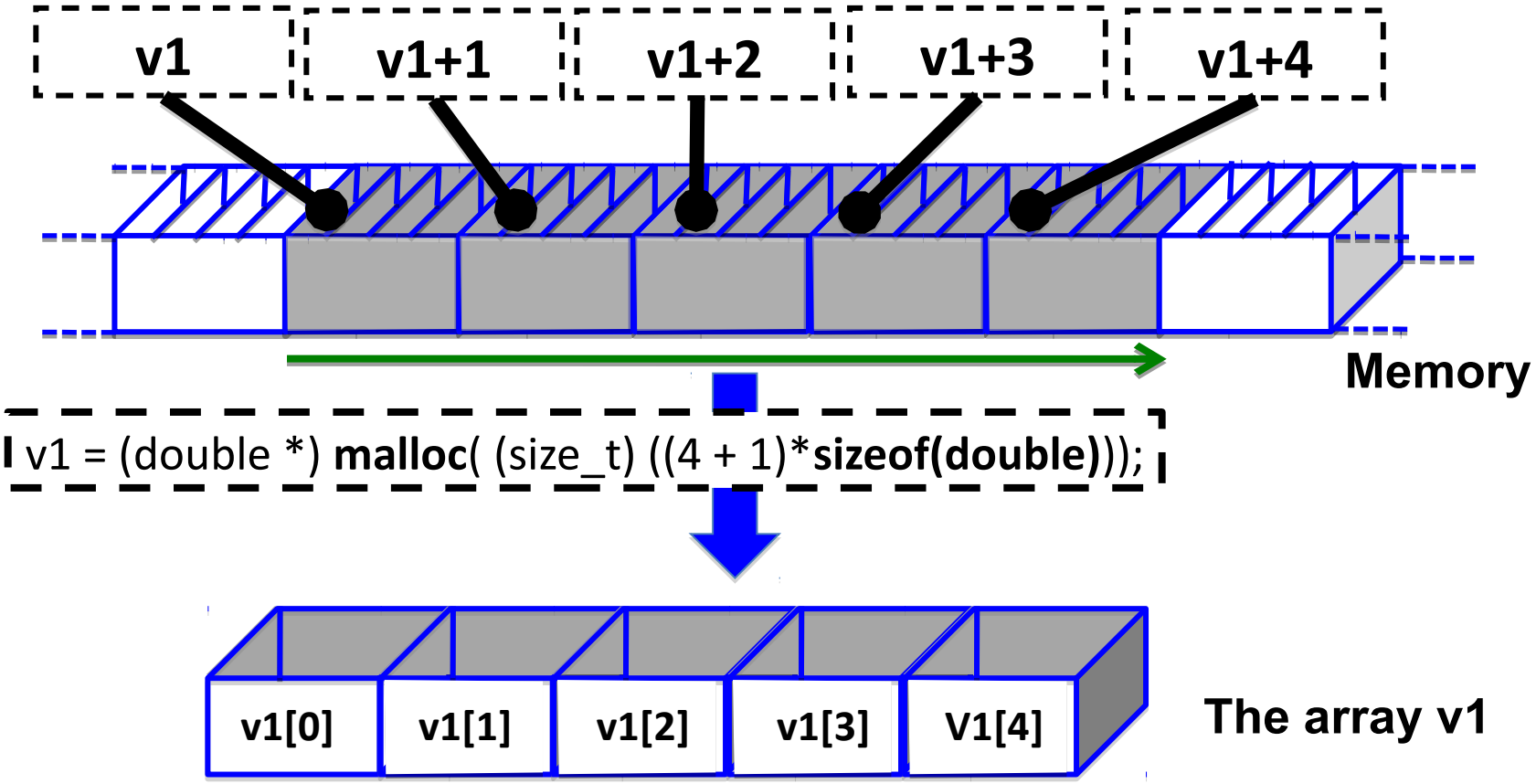
1. **Allocation of memory space to stock (size_v+1) double variables**
2. **malloc() returns the value of address of the top of the allocated memory**
3. **If failed, malloc() returns NULL**

If you'd like to use it as a function, you need to check 'vector_matrix.c'.

8.4 Array and Pointer: dynamic allocation of memory to array (1)

Check 'vector_matrix.c'

```
double *v1;
v1 = d_vector(4);
```



If you do not want to use v1[0], you can skip this…(useful, but not important).

8.4 Array and Pointer: dynamic allocation of memory to array (2)

The recommended way of the definition of 2-D array (i.e. matrix)

The following is a part of function d_matrix() in 'vector_matrix.c'.

```
double **d_matrix(long size_row, long size_column)
{
1   double **x; //pointer variable, pointing to [pointer variable pointing to double]
    long size_row_P = size_row + 1;
    long size_column_P = size_column + 1;

2   x = (double **) malloc((size_t)  (size_row_P*sizeof(double *)));

3   x[0] = (double *) malloc((size_t) (size_row_P*size_column_P*sizeof(double)));

4   for(i = 1; i < size_row_P; i++) x[i] = x[0] + i*size_column_P;

    return x;

}
```

I would like to explain what happens in the program in a graphical way →.

8.4 Array and Pointer: dynamic allocation of memory to array (2)

Try to generate a 2 x 3 matrix:
size_row_P = 3; size_column_P =2;

$$\begin{bmatrix} x[0][0] & x[0][1] \\ x[1][0] & x[1][1] \\ x[2][0] & x[2][1] \end{bmatrix}$$

Value

*x *(x+1) *(x+2)
x[0] x[1] x[2]

x[0][0] x[0][1] x[1][0] x[1][1] x[2][0]x[2][1]

x[i][j] == *(x[i] + j) == *(*(x + i) +j)

Data Type of Value

pointer to double

double

Address

x x+1 x+2

x[0] x[0] +1 x[1] x[1] +1 x[2] x[2] +1

Memory

2 x = (double **) malloc(3…);

free(x);

5 free_d_matrix(x);

free(x[0]);

3 x[0] = (double *) malloc(3*2 = 6…);

4 x[1] = x[0] + 1*2;
x[2] = x[0] + 2*2;

(pointer operation)

8.5 Homework for using Pointer

To make a main() program, which can calculate the determinant and the inverse matrix if it exists for a 2 x 2 matrix from keyboard input and output to display. You need to use functions in 'vector_matrix.c'.