- Username : class-1
- Password :

- Check VS 2019 whether can use
- We will start our course in 18:30
- we will start demonstrate the exercises at 19:15.
- Do not use scanf_s
- Please make sure the TA has recorded your exercise score here before leaving.

Formosa OJ

link: https://oj.nctu.edu.tw/groups/52/bulletins/

To avoid traffic congestion issues, we will open two submission areas(student ID odd and even).

Time limitation: 10000 ms (10 sec)

One second can execute about 10^8 instructions.

Grading Criteria

11/14	Pointer 2	OJ
11/21	File I/O	Visual Studio
11/28	Structure	OJ
12/5	Multiple files for program project	Visual Studio
12/12	Sorting	OJ
12/19	Final exam	OJ and Visual Studio

We will grade these homework on Visual Studio. If possible, we will also provide an Online Judge, where you can modify the format and check the output for correctness. However, when submitting homework, please ensure they are in the required format.

You need to submit the code to new e3, too(same as before).

Demo

For exercise 1, we will ask you to do a simple modification, and you need to show the results; for other exercises, you need to show the results on OJ.

Remember to upload the code to new e3.

Introduction to Computers and Programming

-- ***POINTER –***

WEEK 10

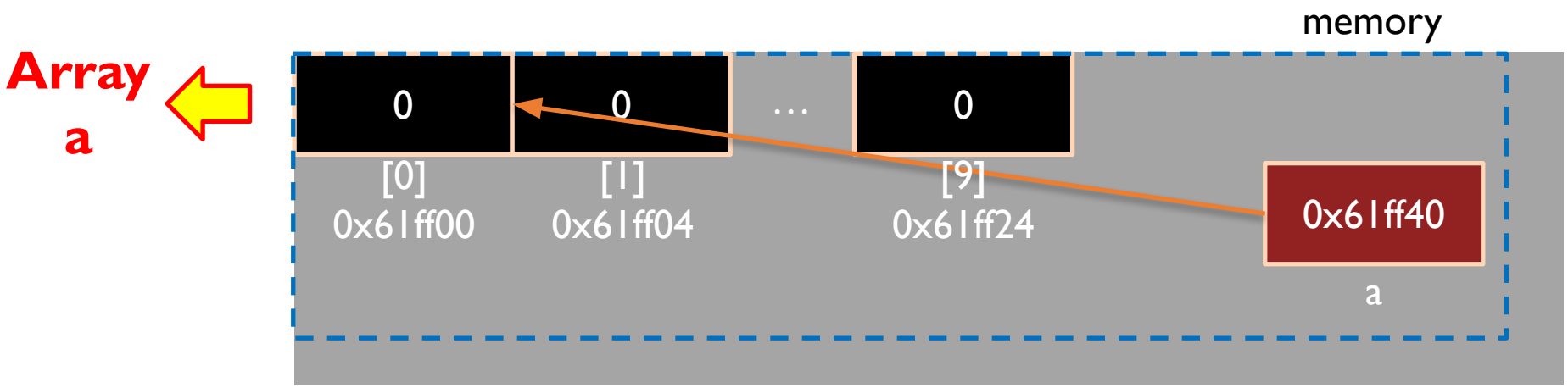
Review: Pointer and Array

Usually we define an array as follows:

```
int a[10];
```

Another way to do it is to declare a pointer and allocate a piece of memory space to it:

```
int *a = NULL;
a = (int*)calloc(10, sizeof(int));
```



Pointer to Pointer

Pointer to Pointer

Just like any other variables, pointers can also be pointed to by other pointers.

To declare a pointer to pointer, we just add an additional *.

```
int **pptr;
```

If you want to declare a pointer to pointer to pointer (3 layers of pointers), then you need 3 stars, and so on.

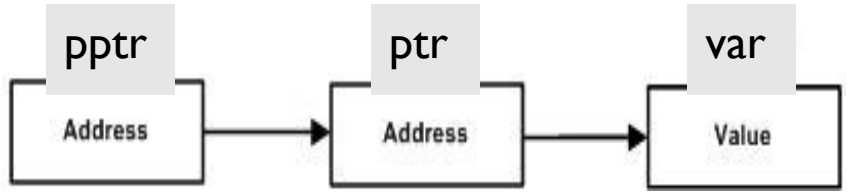
```
int ***a;        //  3 layers
int *****b;      //  5 layers
```

Pointer to Pointer

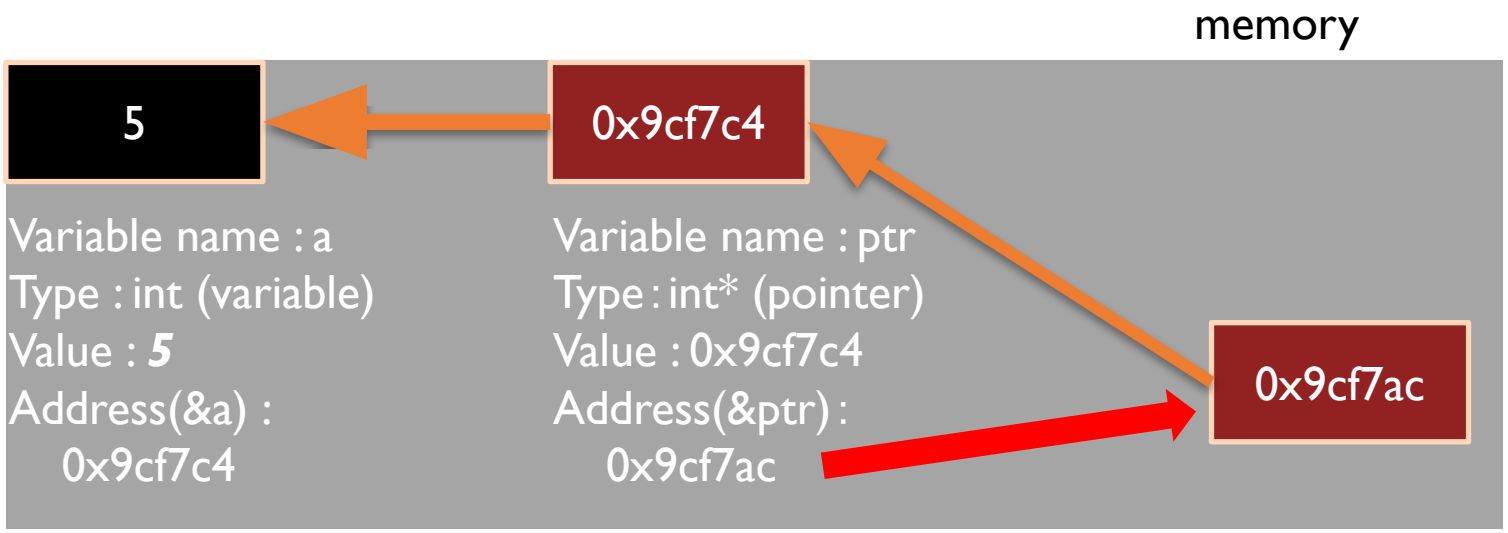
```
int  var;
int  *ptr;
int  **pptr;

var = 123;
ptr = &var;   /* take the address of var */
pptr = &ptr;   /* take the address of ptr using address of operator & */

/* take the value using pptr */
printf("Value of var = %d\n", var );                    //  123
printf("Value available at *ptr = %d\n", *ptr );          //  123
printf("Value available at **pptr = %d\n", **pptr);      //  123
```



Pointer to Pointer



Variable name : pptr
Type : int** (pointer to pointer)
Value : ***0x9cf7ac***
Address(&pptr) :
0x9cf708

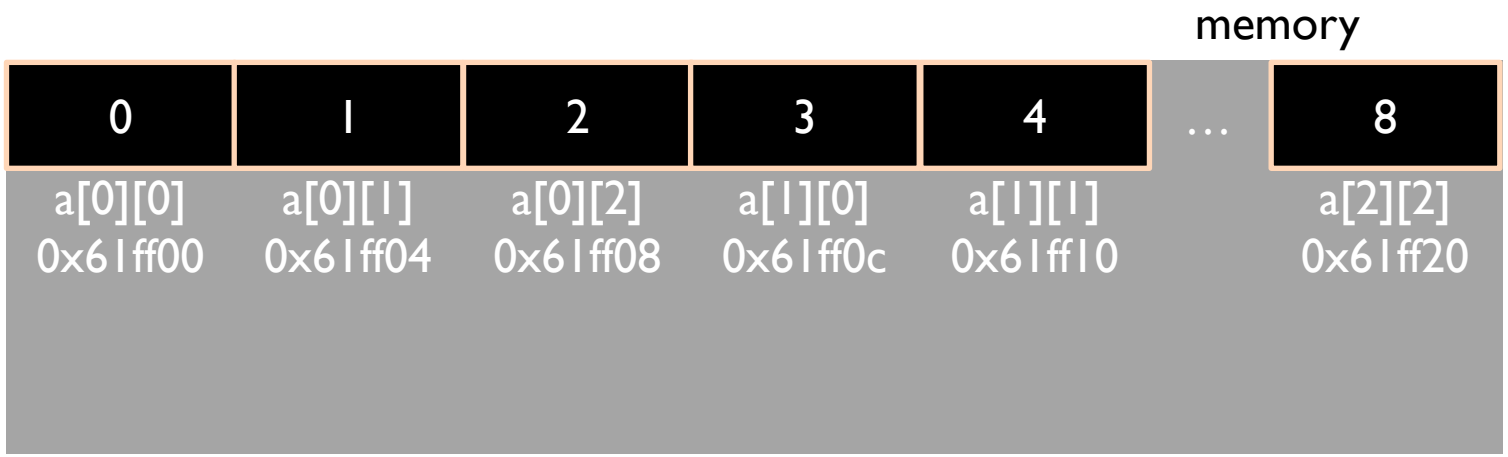
2D-arrays and Pointers

2-D Array

Declare a 2-D array “a”:

```
int a[3][3];
```

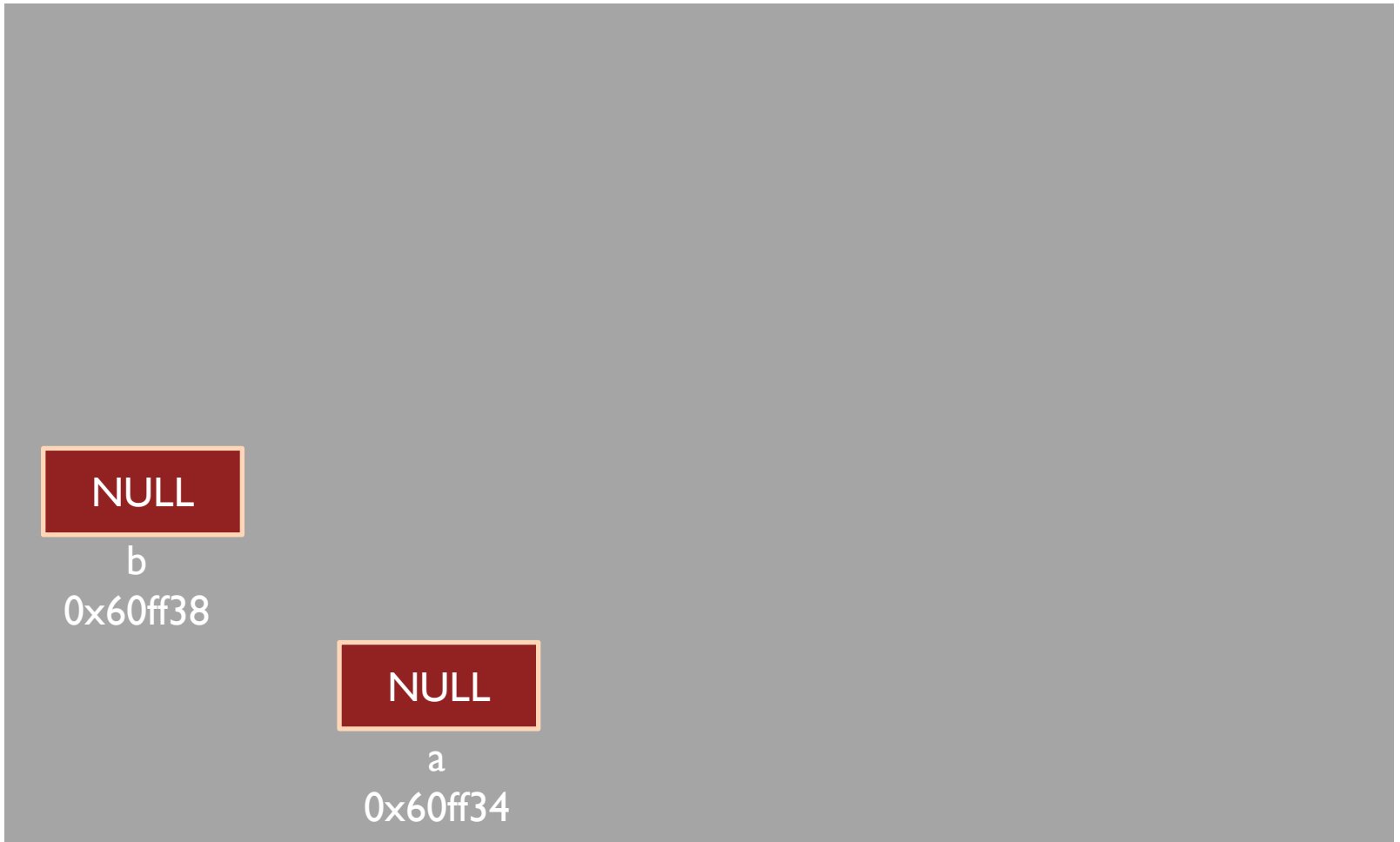
In fact, this array occupies continuous memory space



2-D Array

Or use dynamic memory allocation:

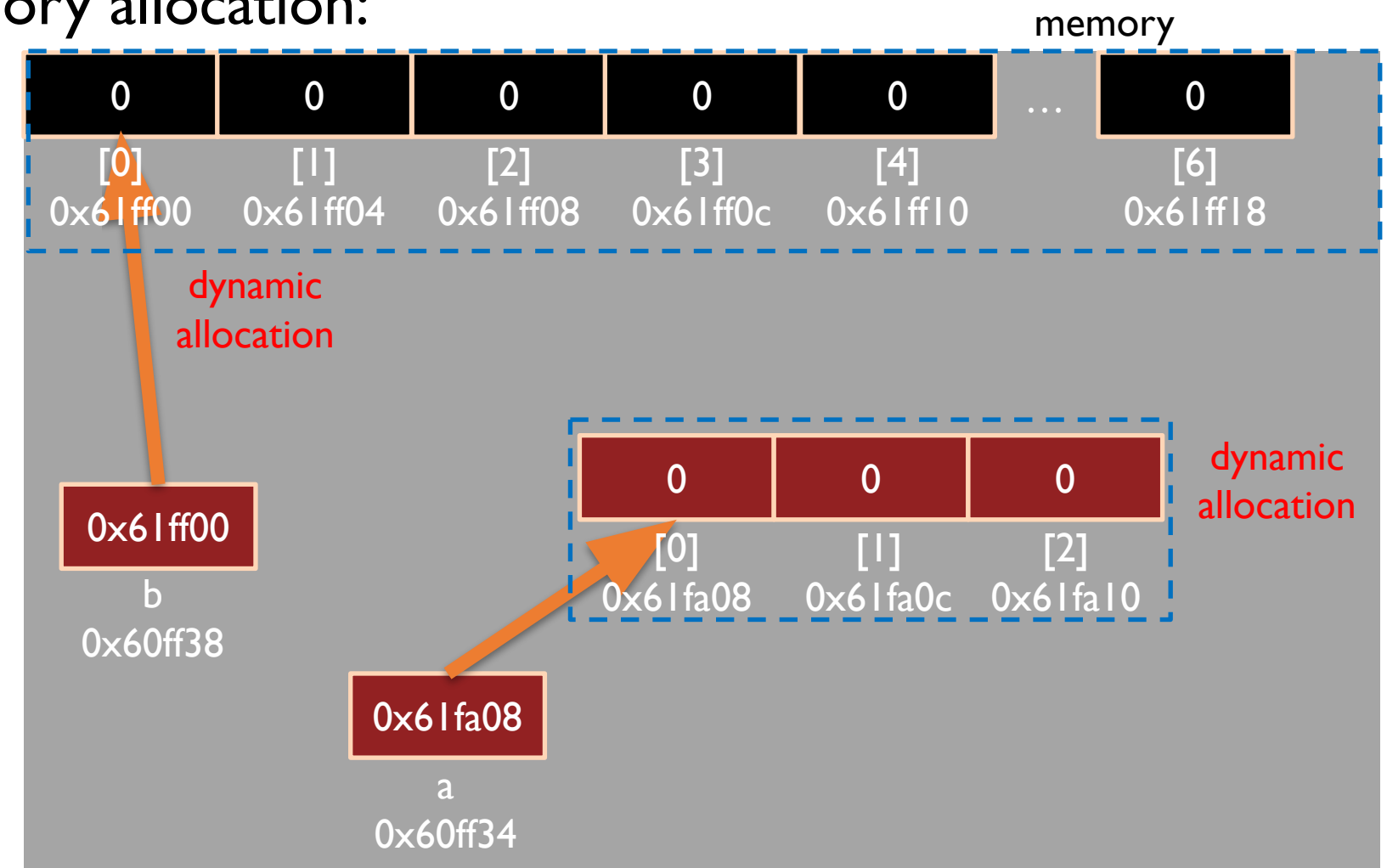
```
int  **a = NULL;
int  *b = NULL;
```



2-D Array

Or use dynamic memory allocation:

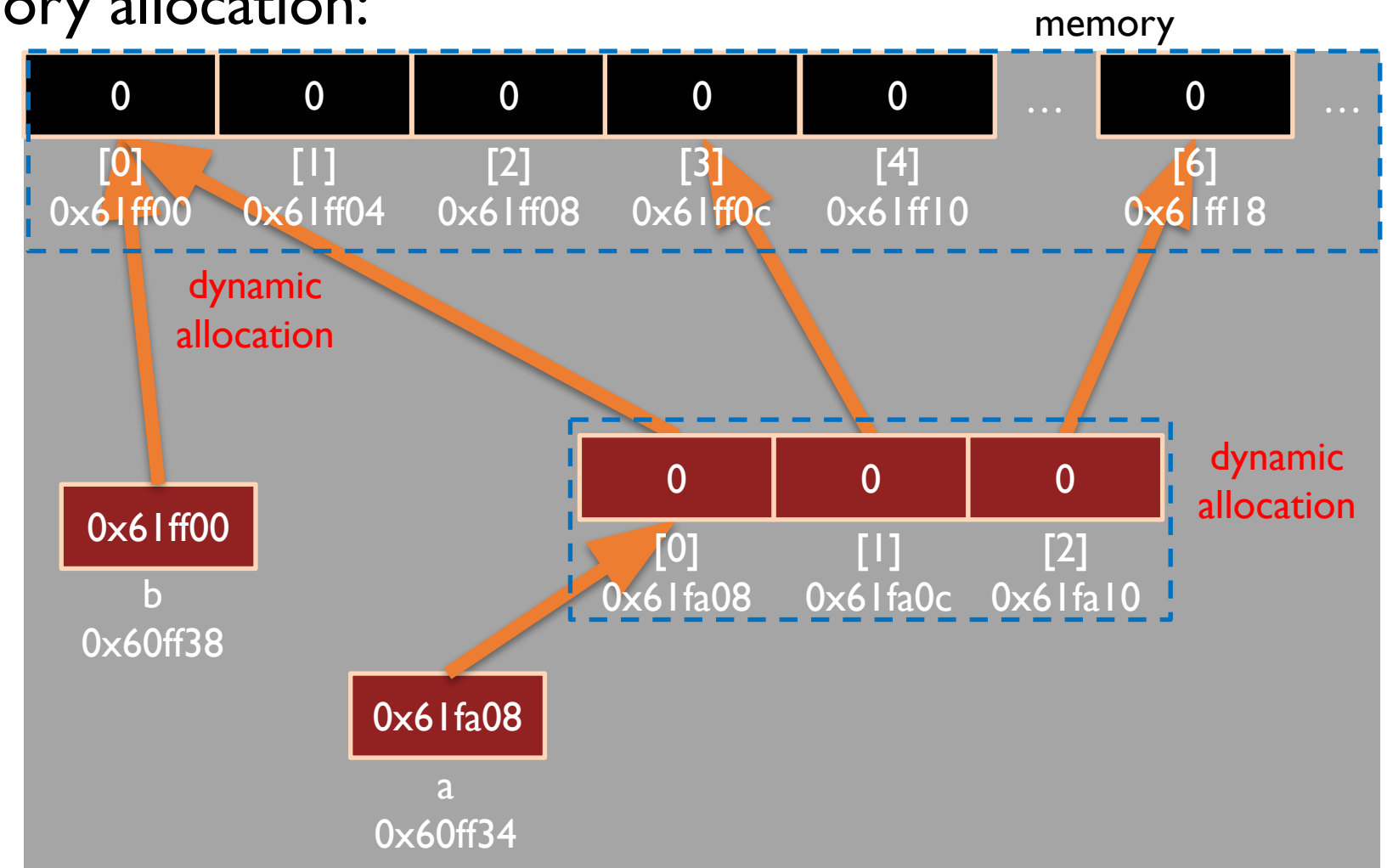
```
a = (int **) calloc( 3, sizeof(int *) )
b = (int *) calloc( 3*3, sizeof(int) )
```



2-D Array

Or use dynamic memory allocation:

```
for ( int i = 0; i < 3; i++) {
  *(a+i) = &b[ 3*i ];
}
```



2-D Array

Or use dynamic memory allocation:

```
int  **a = NULL;
int  *b = NULL;
a = (int **) calloc( 3, sizeof(int *) );    // 3 pointers (pointer array)
b = (int *) calloc( 3*3, sizeof(int) );    // 9 int variables (int array)
for ( int i = 0;  i < 3;  i++) {
  *(a+i) = &b[ 3*i ];     //every pointer corresponds to 3 int variables
}
```

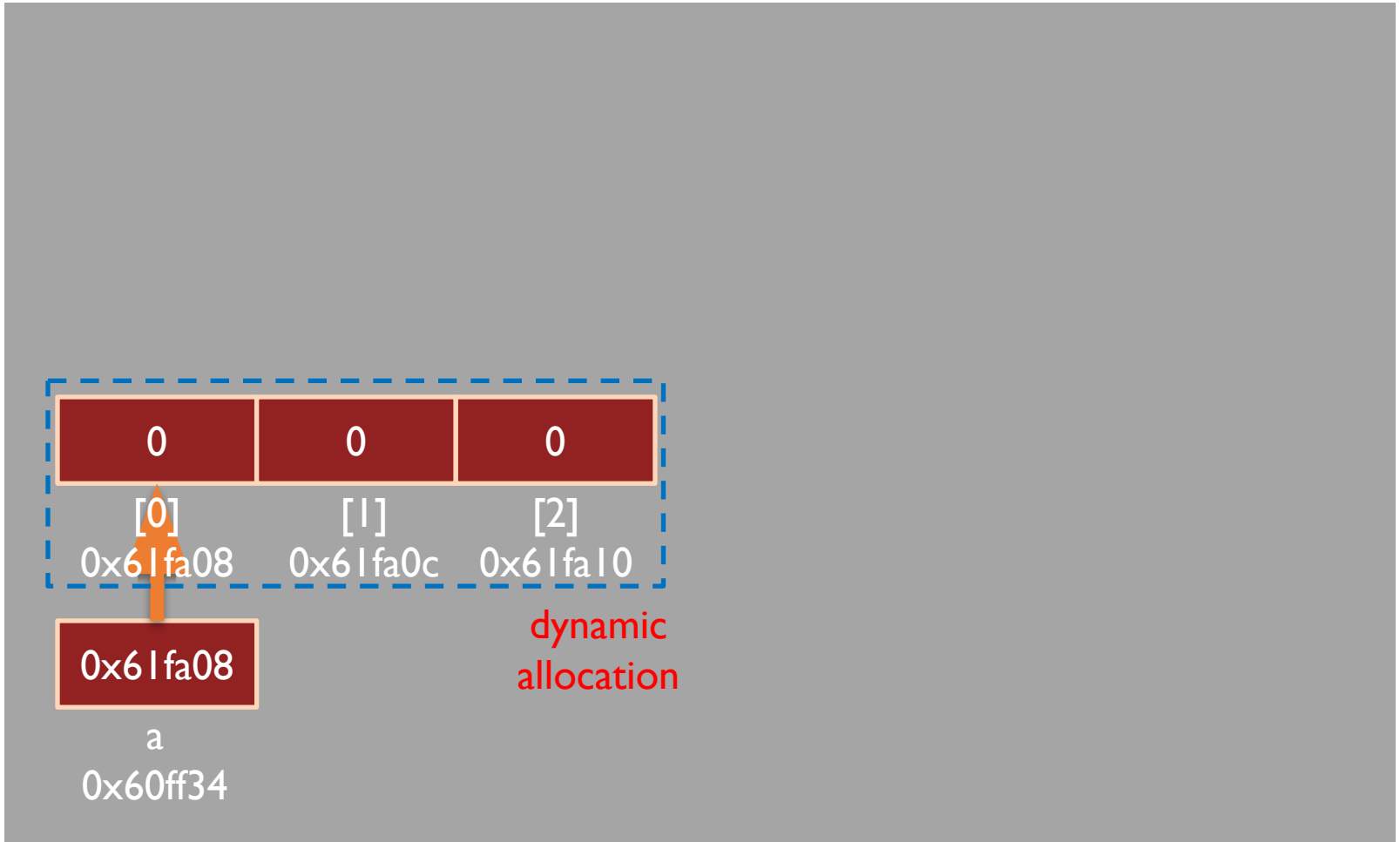
Because the memory space is continuous, all the way below are valid:

```
/* 4 ways to access element in the second row and third column */
a[1][2]
*(*(a+1)+2)
(*a)[5]
*(*a+5)
```

2-D Array

Another way to use dynamic memory allocation:

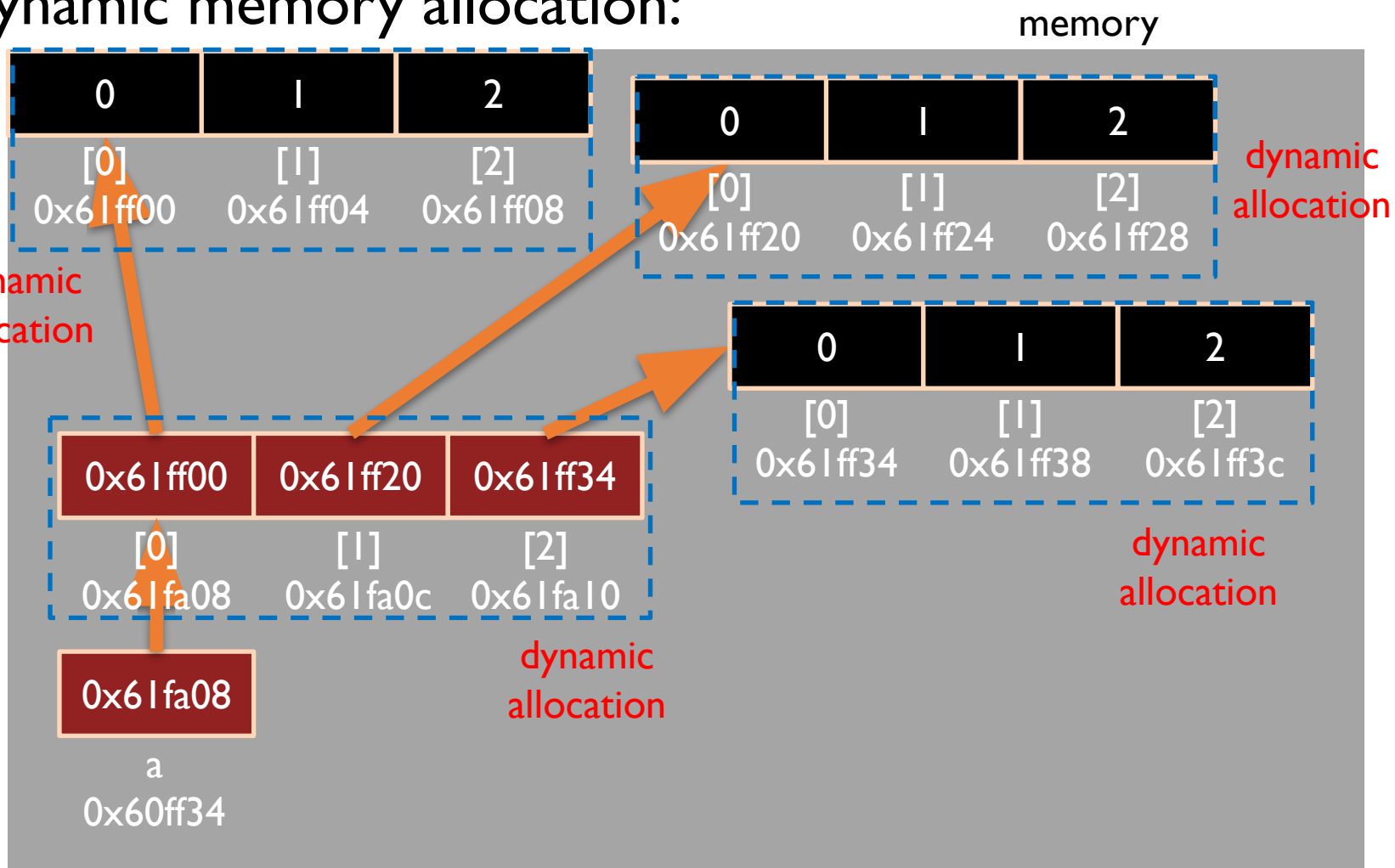
```
int **a = NULL;
a = (int **) calloc(3,sizeof(int *));
```



2-D Array

Another way to use dynamic memory allocation:

```
for ( int i=0 ; i<3 ; i++) {
  *(a+i) = (int *) calloc(3, sizeof(int));
}
```



2-D Array

Another way to use dynamic memory allocation:

```
int **a = NULL;
a = (int **) calloc(3,sizeof(int *));        //3 pointers (pointer array)
for ( int i=0 ; i<3 ; i++) {
  *(a+i) = (int *) calloc(3, sizeof(int));       //each pointer allocates 3 int variables
}
```

Because the memory space is NOT continuous, they cannot be treated as a 1-D array. Only the following usages are valid:

```
/*  ways to access element in the second row and third column */
a[1][2]
*(*(a+1)+2)
```

But because it is not continuous, if the memory usage is tight, this usage is preferred.

Exercise

Exercise 1

- Input a number N, which is the size of an integer pointer array A. Dynamically allocate A with size of N.

- Print out A in each following step.
  1. Input N numbers, which are the sizes of each element in A. They should filled with 0.
  2. Input 2 numbers R and S. Reallocate the Rth array to the size of S. If new size is bigger, fill the new elements with 1.
  3. Input a number N2 (N2 > N). Reallocate array A to the size of N2. Allocate new arrays to it. The size of the new array is 2, and it is filled with 2.
  4. Input a number D. Delete the Dth array. You should also reallocate the array A.

- You should always use free() or realloc() when an array is deleted or resized.

```
input N: 3
step 1:
input N numbers: 5 4 7
0 0 0 0 0
0 0 0 0
0 0 0 0 0 0 0
step 2:
input R and S: 2 9
0 0 0 0 0
0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0
step 3:
input N2: 4
0 0 0 0 0
0 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0
2 2
step 4:
input D: 2
0 0 0 0 0
0 0 0 0 0 0 0
2 2
```

Exercise 2

- First, input m, n, p, which is the dimension of two matrices, A_{m*n}, B_{n*p}. Then, input the two matrices respectively with row-major. Finally, output the multiplication of A and B.

- Note that your code should use the exactly same function signature below

```
int** matrix_multiplication(int **mat1, int **mat2 , int m, int n, int p);
```

Input

- m, n, p (max dimension is 10), two matrices A, B

Output

- Print out the multiplication result of two matrices A and B.
  (It should be a matrix with dimension: m*p.)

Exercise 2

Example:

Input

- m, n, p (max dimension is 10), two matrices A, B

Output

- Print out the multiplication result of two matrices A and B.

(It should be a matrix with dimension: m*p.)

```
6 6 6

1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36

37 38 39 40 41 42
43 44 45 46 47 48
49 50 51 52 53 54
55 56 57 58 59 60
61 62 63 64 65 66
67 68 69 70 71 72
1197 1218 1239 1260 1281 1302
3069 3126 3183 3240 3297 3354
4941 5034 5127 5220 5313 5406
6813 6942 7071 7200 7329 7458
8685 8850 9015 9180 9345 9510
10557 10758 10959 11160 11361 11562
```

Exercise Demo

Exercise 1:

Demo in person

Exercise 2:

Show your OJ result and code to TA

Exercise Submission Format

Format:

- xxxxxxxxx_ex_w10.zip
  - xxxxxxxxx_ex_01.cpp
  - xxxxxxxxx_ex_02.cpp

xxxxxxxxx is your student ID