計算機程式設計

Computer Programming

Arrays

Instructor: 林英嘉

2024/10/14

W6 Slido: #1281345

GitHub repo

Before the course …

- The scoring about this course:
  - Quiz 3% * 15 times (free for three times) = 45%
  - Assignments 5% * 5 times = 25%
  - Midterm exam 10% * 2 times = 20%
  - Final exam 10%

Before the course …

- The scoring about this course:
  - Quiz 3% * 15 times (free for three times) = 45%
  - Assignments 5% * 5 times = 25%
  - Midterm exam 10% * 2 times = 20%
  - Final exam 10%

You will at least get 70% if you take all the quizzes and finish the assignments.
💪💪💪

Before the course …

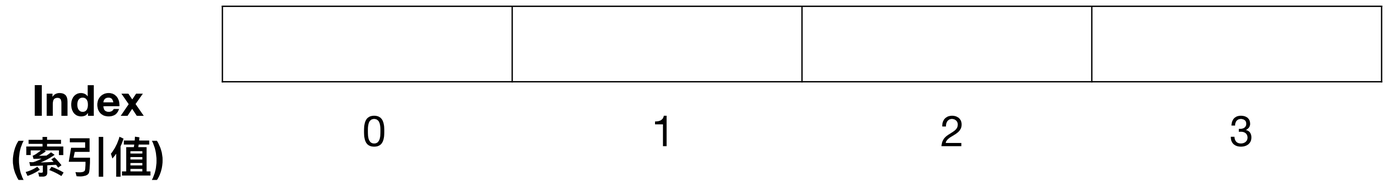
- You can download the Midterm exam questions and answers via
  - https://github.com/mcps5601/C-course-materials

Outline

- Int Arrays
  - Uni-dimensional arrays
  - Multi-dimensional arrays
- Memory Address of Arrays
  - The concept of memory address

What are Arrays?

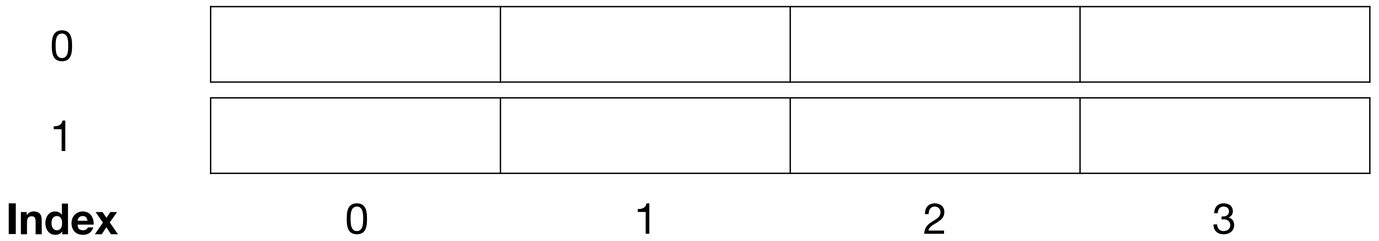
- Arrays: 陣列
- An array is a data structure containing a number of data values, all of which have the same type.



Arrays in different dimensions

One-dimensional Array

Two-dimensional Array



Arrays in different dimensions

One-dimensional Array

Index 0 1 2 3

Two-dimensional Array

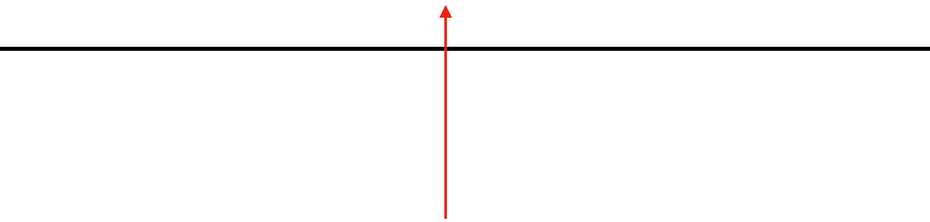
0

1

Index 0 1 2 3

Declaration of Arrays

```
data_type array_name[num_elements];
```



Declaration of Arrays

```
data_type array_name[num_elements];
```

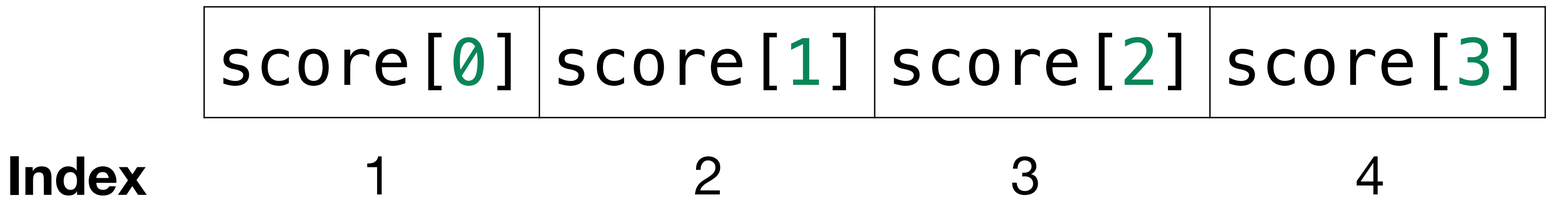

the number of elements in the array

```
int score[4]; // 4 int elements in the array `score`
float temp[7]; // 7 float elements in the array `temp`
```

Array Subscripting

To access a value in an array, we can perform array subscripting (or called array indexing).

使用索引取得陣列中的元素



Array Subscripting

To access a value in an array, we can perform array subscripting (or called array indexing).

使用索引取得陣列中的元素

```
int score[4];
```

score[0]	score[1]	score[2]	score[3]
Index	1	2	3	4

Assigning values to an array (Example 1)

C-course-materials/04-Arrays/assigning_val_ex1.c

```c
#include <stdio.h>
int main(){
    int score[4]; // 4 integer elements in the array `score`
    score[0] = 80;
    score[1] = 85;
    score[2] = 90;
    score[3] = 100;

    for (int i = 0; i < 4; i++){
        printf("Value: %d (index: %d)\n", score[i], i);
    }
}
```

Assigning values to an array (Example 2)

C-course-materials/04-Arrays/assigning_val_ex2.c

Input

```
80 85 90 100
```

```c
#include <stdio.h>
int main(){
    int score[4];
    int i = 0; // i: index

    while (i < 4) {
        scanf("%d", &score[i]);
        i++;
    }
    for (int i = 0; i < 4; i++){
        printf("Value: %d (index: %d)\n", score[i], i);
    }
}
```

Array Initialization

- We can use braces to initialize the values for an array.

```
int score[4] = {80, 85, 90, 100};
```

Array Initialization

- We can use braces to initialize the values for an array.

```
int score[4] = {80, 85, 90, 100};
```

Initializer

Key Points When Initializing arrays (1)

- When an initializer is present, the **length** of the array **can be omitted**.

```
int score[] = {80, 85, 90, 100};
```

Key Points When Initializing arrays (2)

C-course-materials/04-Arrays/shorter_initializer.c

- When the initializer is shorter than the array, zero values will be appended.

```c
#include <stdio.h>
int main(){
    int score[4] = {80, 85, 90};
    for (int i = 0; i < 4; i++){
        printf("Value: %d (index: %d)\n", score[i], i);
    }
}
```

Output

```
Value: 80 (index: 0)
Value: 85 (index: 1)
Value: 90 (index: 2)
Value: 0 (index: 3)
```

Key Points When Initializing arrays (2)

- When the initializer is shorter than the array, zero values will be appended.
  - This process is also called zero-initialization.
- Advantage: We can easily initialize a big array with all zeros.

```
int score[10000] = {};
int score[10000] = {0};
```


At least one value should be placed in an initializer for earlier C standards like C90 or C99
(Default C11 in gcc 8.3)

Key Points When Initializing arrays (3)

C-course-materials/04-Arrays/longer_initializer.c

- An initializer **should not be longer** than its array.

```c
#include <stdio.h>
int main(){
    int score[3] = {80, 85, 90, 100};
    for (int i = 0; i < 4; i++){
        printf("Value: %d (index: %d)\n", score[i], i);
    }
}
```

- C does not check the boundary of an array, so this code will not cause a compilation error.
- In this case, the last one is the garbage value left in the memory.

How to get the array length?

- We use **score[0]** to get the size of an element in an array.

```
sizeof(array)/sizeof(array[0]));
```

Get the size of the entire array

Get the size of an element in the array

Array

An element

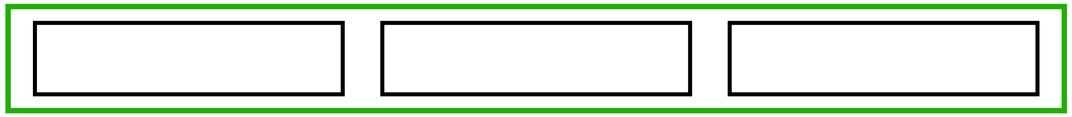
How to get the array length?

- We use **score[0]** to get the size of an element in an array.

```
sizeof(array)/sizeof(array[0]));
```

Get the size of the entire array

Get the size of an element in the array



Array

An element

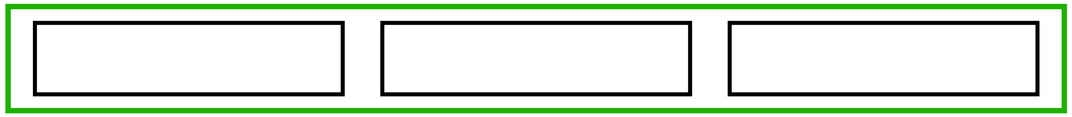
How to get the array length?

- We use **score[0]** to get the size of an element in an array.

```
sizeof(array)/sizeof(array[0]));
```

Get the size of the entire array

Get the size of an element in the array



- Array length = Number of elements

Array

An element

Using sizeof for arrays

C-course-materials/04-Arrays/sizeof_array.c

- We use **score[0]** to get the size of an element in an array.

```c
#include <stdio.h>
int main(){
    int score[4] = {80, 85, 90, 100};
    printf("Size of the int array: %d bytes\n", sizeof(score));
    printf("Array length: %d\n", sizeof(score)/sizeof(score[0]));

    float f_score[4];
    printf("Size of the float array: %d bytes\n", sizeof(f_score));
    printf("Array length: %d\n", sizeof(f_score)/sizeof(f_score[0]));

    double d_score[4];
    printf("Size of the double array: %d bytes\n", sizeof(d_score));
    printf("Array length: %d\n", sizeof(d_score)/sizeof(d_score[0]));
}
```

資料類型比較

	大小 (Byte)*	Specifier	數值範圍
`int`	4	**%d**	-2,147,483,648 到 2,147,483,647 (範圍2的32次方)
`char`	1	**%c**	-128 到 127 或 0 到 255 （取決於是否有符號）
`float`	4	**%f**	約 1.2E-38 到 3.4E+38，精度約 6 位十進制之小數
`double`	**8**	**%lf**	約 2.2E-308 到 1.7E+308，精度約 15-16 位十進制之小數

*以64-bit系統為例

Example Problem: Fibonacci Numbers

Write a program that accepts an input N and prints the N-th Fibonacci number

$$fib(i) = \begin{cases} 0, & i = 0 \\ 1, & i = 1 \\ fib(i-1) + fib(i-2), & i > 1 \end{cases}$$

$fib(i)$	0	1	1	2	3	5	8	…
Index	0	1	2	3	4	5	6	

Problem: Fibonacci Numbers

C-course-materials/04-Arrays/fibonacci.c

- Print the N-th Fibonacci Number

```c
#include <stdio.h>
int main(){
    int fib[100];
    int n;
    scanf("%d", &n);
    fib[0] = 0;
    fib[1] = 1;
    for (int i = 2; i < n; i++){
        fib[i] = fib[i-1] + fib[i-2];
        printf("%d\n", fib[i]);
    }
    printf("The n-th number is: %d\n", fib[n-1]);
}
```

Problem: Checking a Number for Repeated Digits

Write a program that accepts an input **positive integer**
and checks if there is a repeated digit inside it.

Input	`28212`
Output	`Repeated digit found!!`

Input	`12345`
Output	`No repeated digit!!`

Problem: Checking a Number for Repeated Digits

C-course-materials/04-Arrays/repeated_digits.c

```c
#include <stdio.h>
int main() {
    int digit_seen[10] = {0};
    int input_number, digit;
    scanf("%d", &input_number);

    while (input_number > 0) {
        digit = input_number % 10; // Extract the last digit
        if (digit_seen[digit] == 1) {
            printf("Repeated digit\n");
            break;
        }
        else
            digit_seen[digit] = 1; // Mark the digit as seen
        input_number /= 10; // Remove the last digit from the input number
    }
    if (input_number == 0)
        printf("No repeated digit\n");
}
```

Problem: Bubble Sort

Write a program that can print **a sequence in an increasingly sorted manner**.

```
int unsorted[5] = {26, 5, 81, 7, 63};
```

Output

```
5 7 26 63 81
```

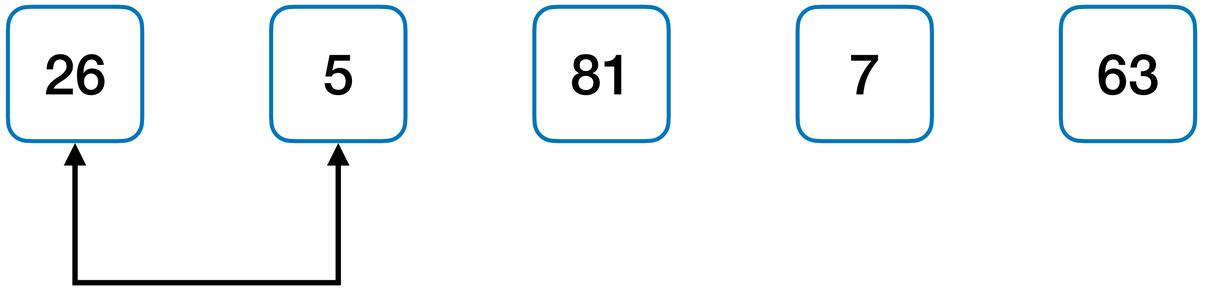
```
int unsorted[5] = {5, 4, 3, 2, 1};
```

Output

```
1 2 3 4 5
```

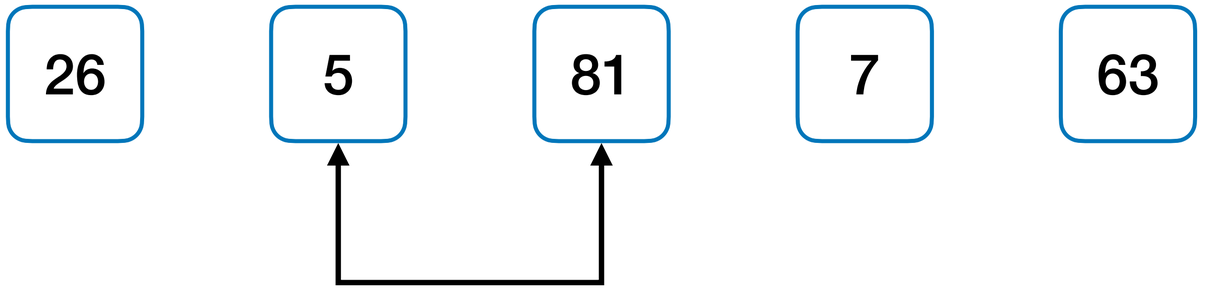
Problem: Bubble Sort (illustration)

- We need to sort the sequence for an increasing order. (small to big)



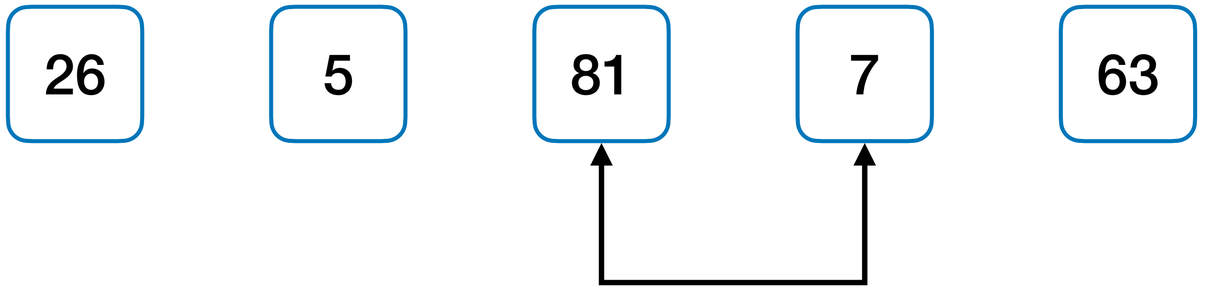
Problem: Bubble Sort (illustration)

- We need to sort the sequence for an increasing order. (small to big)



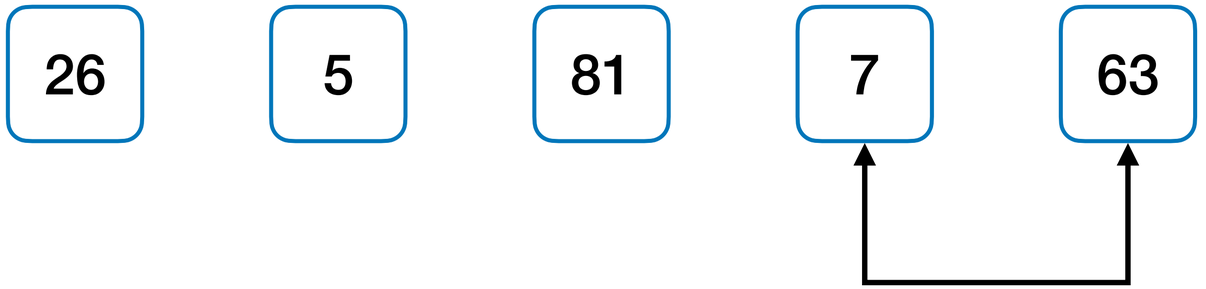
Problem: Bubble Sort (illustration)

- We need to sort the sequence for an increasing order. (small to big)



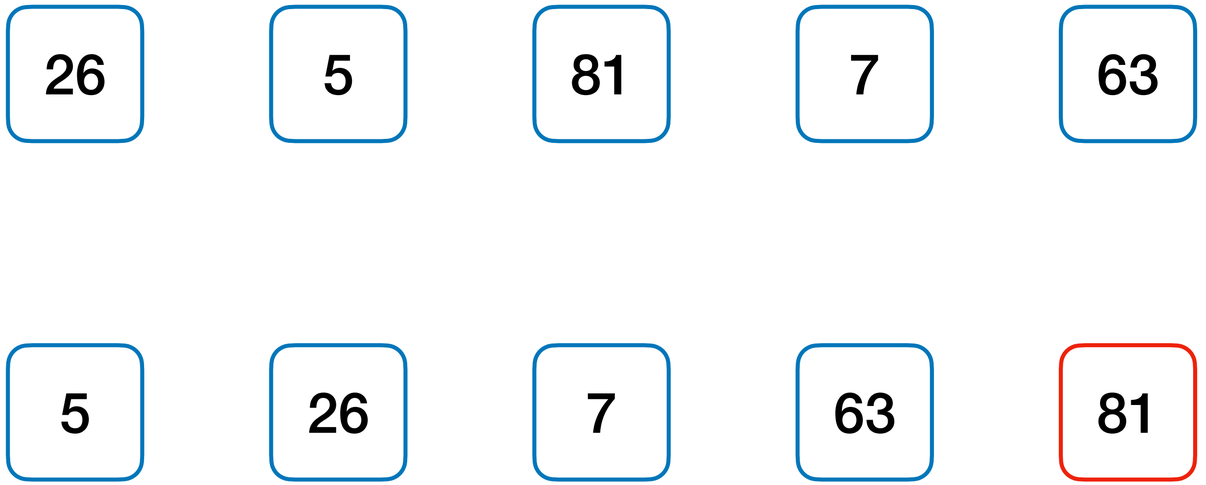
Problem: Bubble Sort (illustration)

- We need to sort the sequence for an increasing order. (small to big)



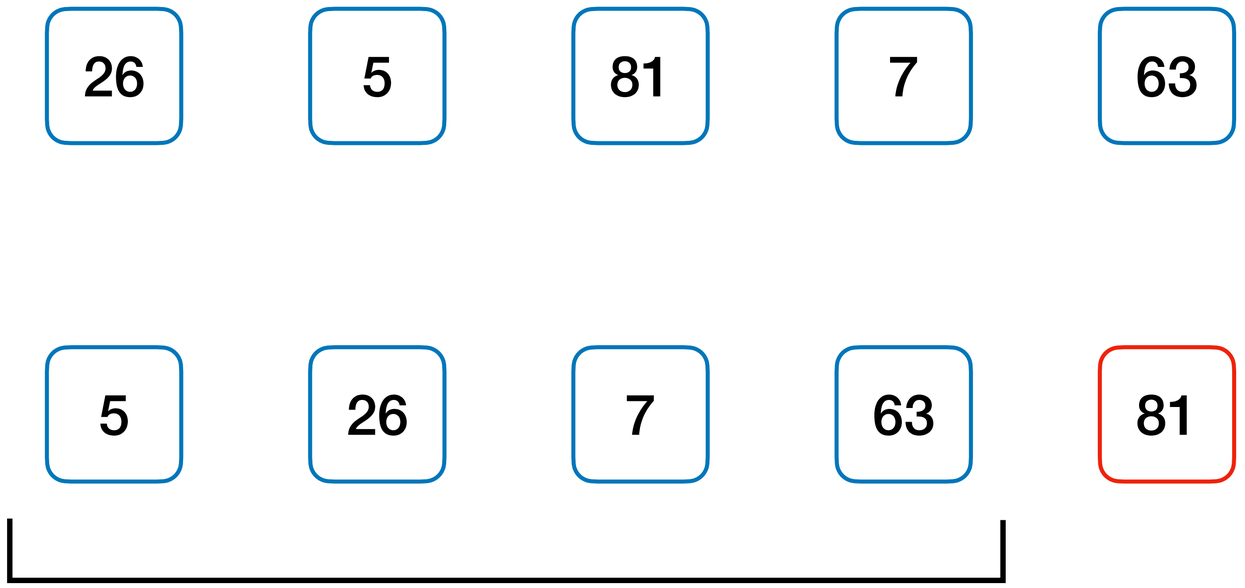
Problem: Bubble Sort (illustration)

- We need to sort the sequence for an increasing order. (small to big)



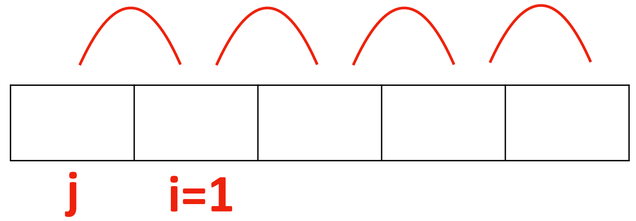
Problem: Bubble Sort (illustration)

- We need to sort the sequence for an increasing order. (small to big)



Next: Only compare the items except the most right one

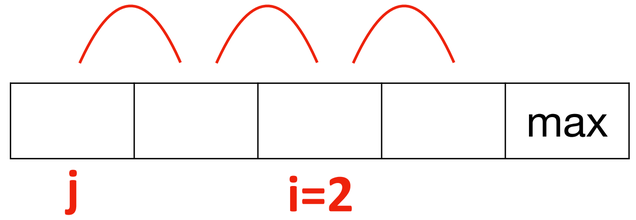
Problem: Bubble Sort (code)



C-course-materials/04-Arrays/bubble_sort.c

```c
#include <stdio.h>
int main(){
    int unsorted[5] = {26, 5, 81, 7, 63};
    int temp = 0;
    // start bubble sort
    for (int i = 1; i < 5; i++){
        for (int j = 0; j < 5 - i; j++){
            if (unsorted[j] > unsorted[j+1]){
                // swap
                temp = unsorted[j];
                unsorted[j] = unsorted[j+1];
                unsorted[j+1] = temp;
            }
        }
    }
    for (int i = 0; i < 5; i++){
        printf("%d\n", unsorted[i]);
    }
}
```

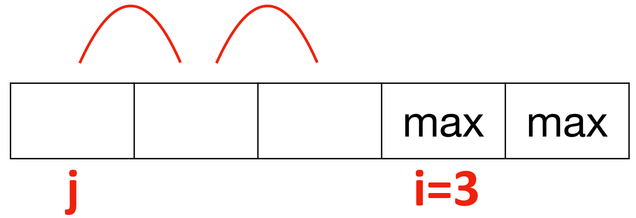
Problem: Bubble Sort (code)



C-course-materials/04-Arrays/bubble_sort.c

```c
#include <stdio.h>
int main(){
    int unsorted[5] = {26, 5, 81, 7, 63};
    int temp = 0;
    // start bubble sort
    for (int i = 1; i < 5; i++){
        for (int j = 0; j < 5 - i; j++){
            if (unsorted[j] > unsorted[j+1]){
                // swap
                temp = unsorted[j];
                unsorted[j] = unsorted[j+1];
                unsorted[j+1] = temp;
            }
        }
    }
    for (int i = 0; i < 5; i++){
        printf("%d\n", unsorted[i]);
    }
}
```

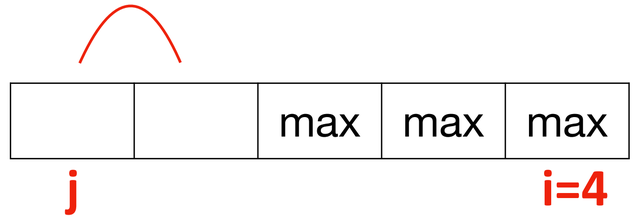
Problem: Bubble Sort (code)



C-course-materials/04-Arrays/bubble_sort.c

```c
#include <stdio.h>
int main(){
    int unsorted[5] = {26, 5, 81, 7, 63};
    int temp = 0;
    // start bubble sort
    for (int i = 1; i < 5; i++){
        for (int j = 0; j < 5 - i; j++){
            if (unsorted[j] > unsorted[j+1]){
                // swap
                temp = unsorted[j];
                unsorted[j] = unsorted[j+1];
                unsorted[j+1] = temp;
            }
        }
    }
    for (int i = 0; i < 5; i++){
        printf("%d\n", unsorted[i]);
    }
}
```

Problem: Bubble Sort (code)



C-course-materials/04-Arrays/bubble_sort.c

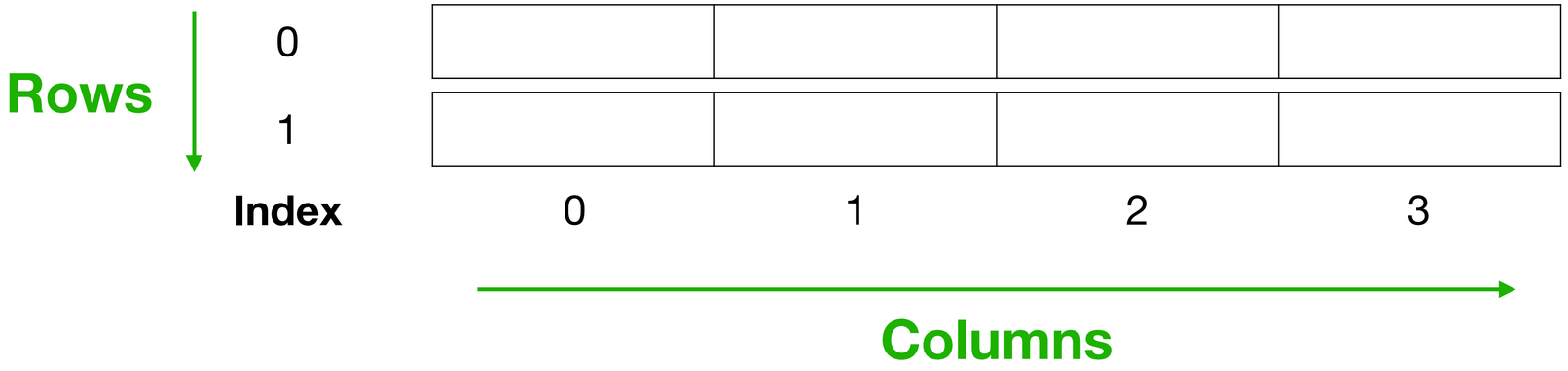
```c
#include <stdio.h>
int main(){
    int unsorted[5] = {26, 5, 81, 7, 63};
    int temp = 0;
    // start bubble sort
    for (int i = 1; i < 5; i++){
        for (int j = 0; j < 5 - i; j++){
            if (unsorted[j] > unsorted[j+1]){
                // swap
                temp = unsorted[j];
                unsorted[j] = unsorted[j+1];
                unsorted[j+1] = temp;
            }
        }
    }
    for (int i = 0; i < 5; i++){
        printf("%d\n", unsorted[i]);
    }
}
```

Multi-dimensional arrays

Multidimensional Arrays

- Take a two-dimensional array as an example:

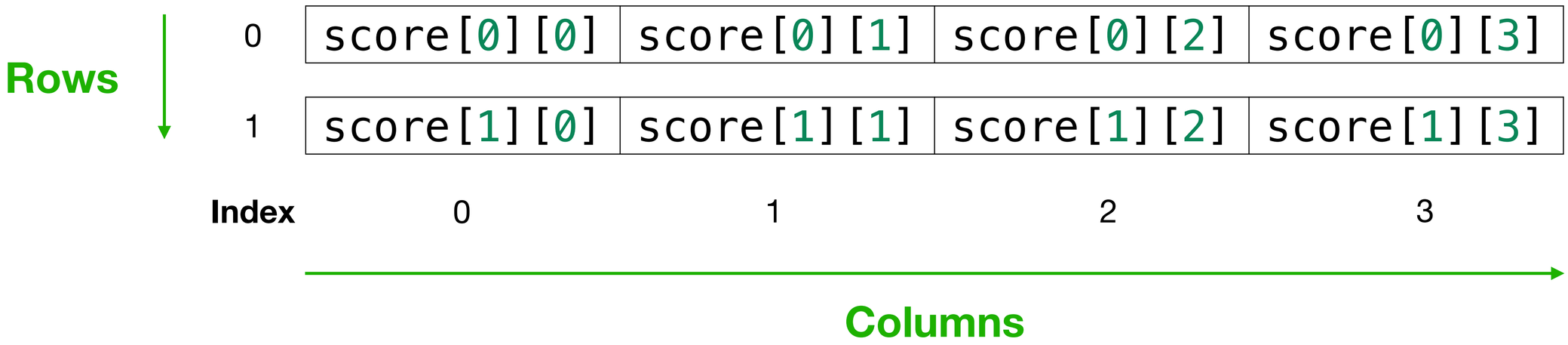
```
int score[2][4];
```




```
data_type array_name[num_rows][num_columns];
```

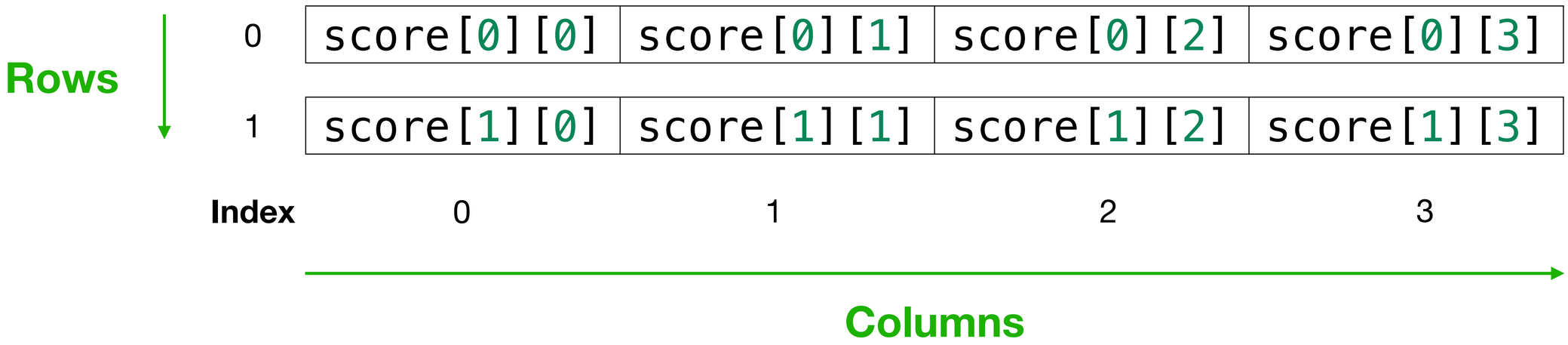
dimensions

Array Subscripting (Two-dimensional)



Array Subscripting (Two-dimensional)

```
int score[2][4];
```



Array Initialization (Two-dimensional)

- We can use nested braces to initialize the values for a 2-D array.

```
int score[2][4] = {
    {80, 85, 90, 100},
    {60, 65, 70, 100}
};
```

Array Initialization (Two-dimensional)

- We can use nested braces to initialize the values for a 2-D array.

```
int score[2][4] = {
    {80, 85, 90, 100},
    {60, 65, 70, 100}
};
```

Corresponding to rows (2)

Array Initialization (Two-dimensional)

- We can use nested braces to initialize the values for a 2-D array.

```
int score[2][4] = {
    {80, 85, 90, 100},
    {60, 65, 70, 100}
};
```

Corresponding to rows (2)

Corresponding to columns (4)

You can omit one of the dimensions

- Same results can be obtained from the following code.

```
int score[][4] = {
    {80, 85, 90, 100},
    {60, 65, 70, 100}
};
```

```
int score[2][] = {
    {80, 85, 90, 100},
    {60, 65, 70, 100}
};
```

- Omit both dimensions will raise a compilation error.

Assigning values to an array (while, scanf)

Input `80 85 90 100 60 65 70 100`


```c
#include <stdio.h>
int main(){
    int score[2][4];
    int i = 0, row = 0, col = 0;

    while (i < 8) {
        scanf("%d", &score[row][col]);
        i++;  // index increment
        col++;
        if (i % 4 == 0){
            row++;
            col = 0;
        }
    }
```

Assigning values to an array (while, scanf)


```
    for (row = 0; row < 2; row++){
        for (col = 0; col < 4; col++){
            printf("%d at (%d, %d)\n", score[row][col], row, col);
        }
    }
}
```

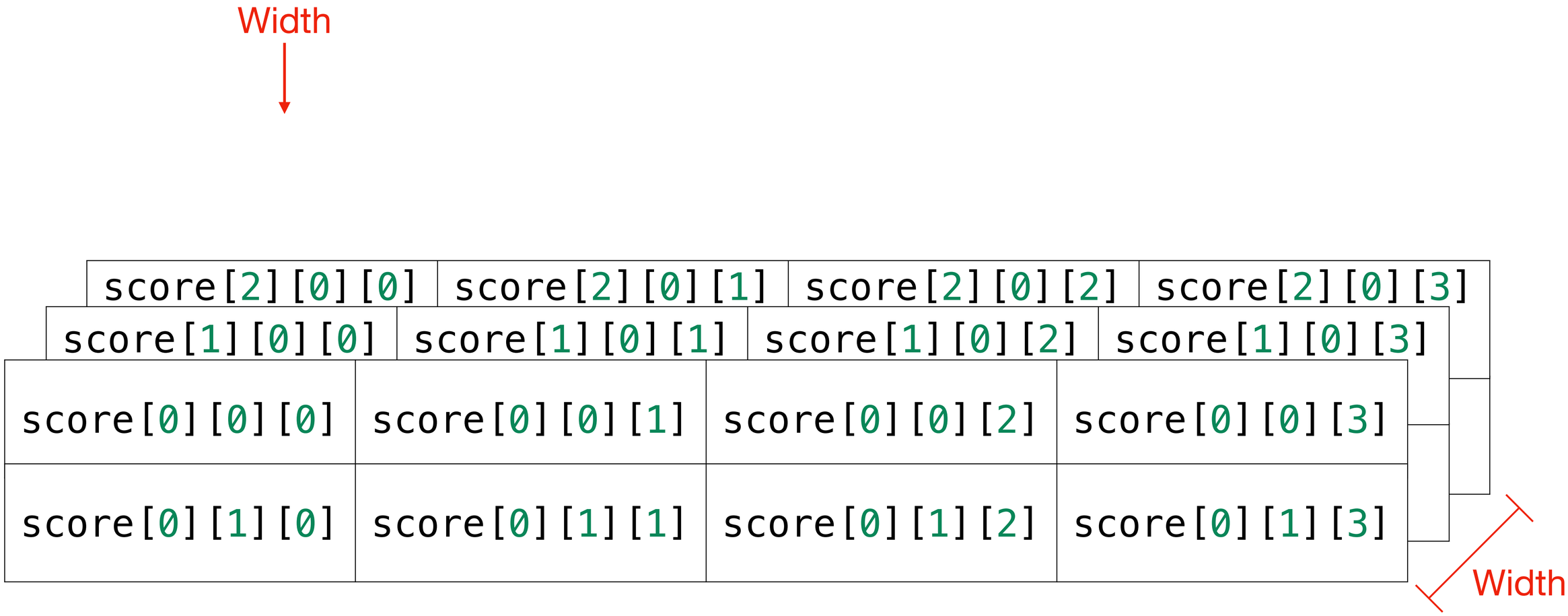
Assigning values to an array (for, scanf)

Input

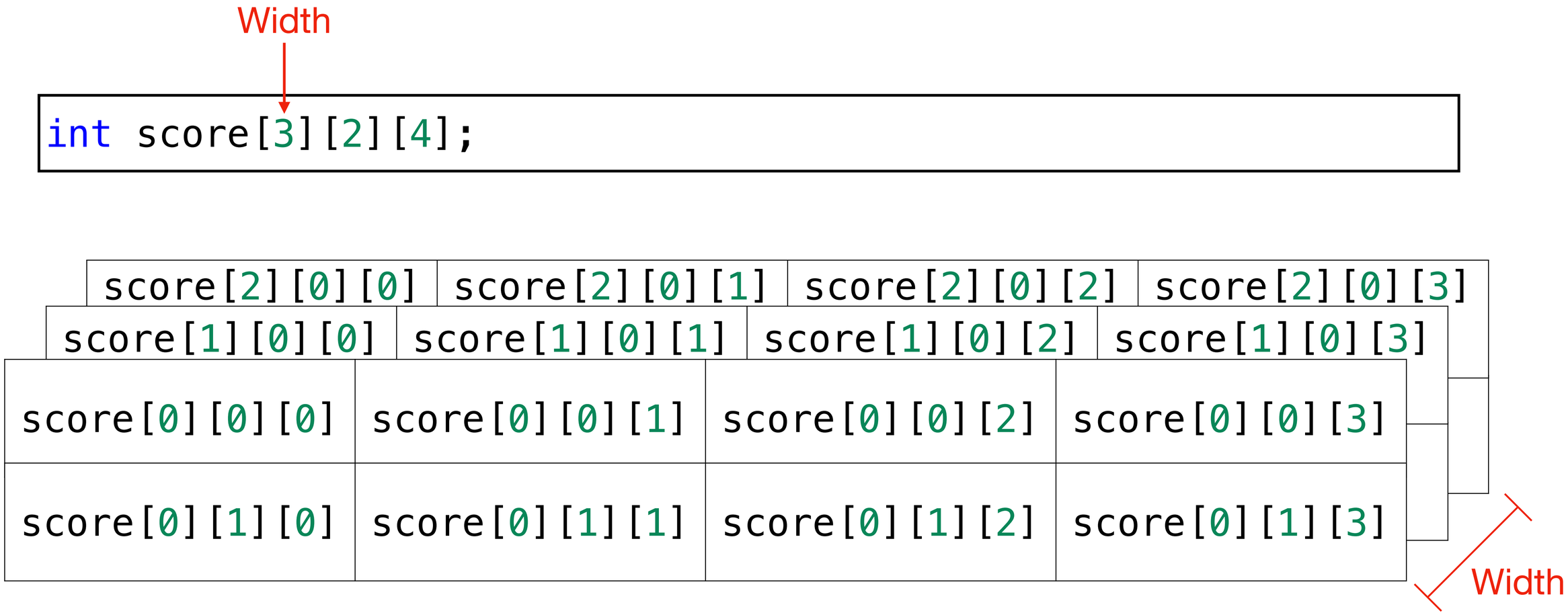
```
80 85 90 100 60 65 70 100
```

```c
#include <stdio.h>
int main(){
    int score[2][4];
    int i = 0, row = 0, col = 0;

    for (row = 0; row < 2; row++){
        for (col = 0; col < 4; col++){
            scanf("%d", &score[row][col]);
        }
    }
    for (row = 0; row < 2; row++){
        for (col = 0; col < 4; col++){
            printf("%d at (%d, %d)\n", score[row][col], row, col);
        }
    }
}
```

Three-dimensional Arrays



Three-dimensional Arrays



Array Initialization (Three-dimensional)

- We can also use nested braces to initialize the values for a 3-D array.
- This is a **2-D** array initialization:

```
int score[2][4] = {
    {80, 85, 90, 100},
    {60, 65, 70, 100}
};
```

Array Initialization (Three-dimensional)

- We can also use nested braces to initialize the values for a 3-D array.
- This is a **3-D** array initialization (in this case, three times of the 2-D array):

```
int score[3][2][4] = {{
    {80, 85, 90, 100},
    {60, 65, 70, 100}
},
{
    {80, 85, 90, 100},
    {60, 65, 70, 100}
},
{
    {80, 85, 90, 100},
    {60, 65, 70, 100}
}};
```

Observation for the shape of a 3-D array

C-course-materials/04-Arrays/observe_shape_3D.c

- Get the number of all elements of the array

```
int total_elements = sizeof(score) / sizeof(score[0][0][0]);
```

- Get the size of the first dimension

```
int dim1 = sizeof(score) / sizeof(score[0]);
```

- Get the size of the second dimension

```
int dim2 = sizeof(score[0]) / sizeof(score[0][0]);
```

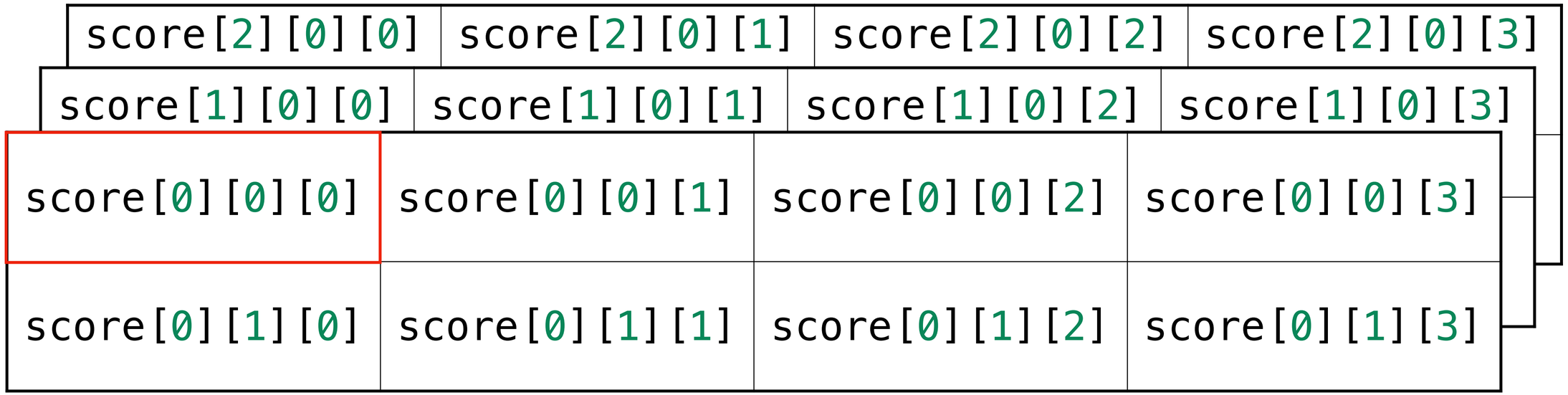
- Get the size of the third dimension

```
int dim3 = sizeof(score[0][0]) / sizeof(score[0][0][0]);
```

Observation for the shape of a 3-D array

- Get the number of all elements of the array

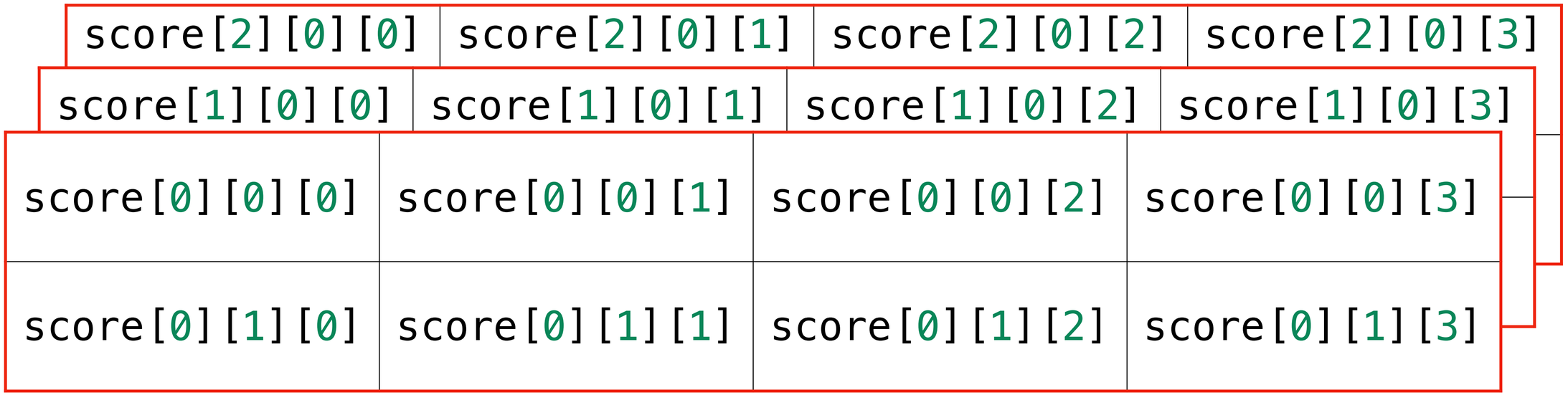
```
int total_elements = sizeof(score) / sizeof(score[0][0][0]);
```



Observation for the shape of a 3-D array

- Get the size of the first dimension [3]

```
int dim1 = sizeof(score) / sizeof(score[0]);
```



Observation for the shape of a 3-D array

- Get the size of the second dimension [2]

```
int dim2 = sizeof(score[0]) / sizeof(score[0][0]);
```

score[0][0][0]	score[0][0][1]	score[0][0][2]	score[0][0][3]
score[0][1][0]	score[0][1][1]	score[0][1][2]	score[0][1][3]

Observation for the shape of a 3-D array

- Get the size of the third dimension [4]

```
int dim3 = sizeof(score[0][0]) / sizeof(score[0][0][0]);
```

score[0][0][0]	score[0][0][1]	score[0][0][2]	score[0][0][3]

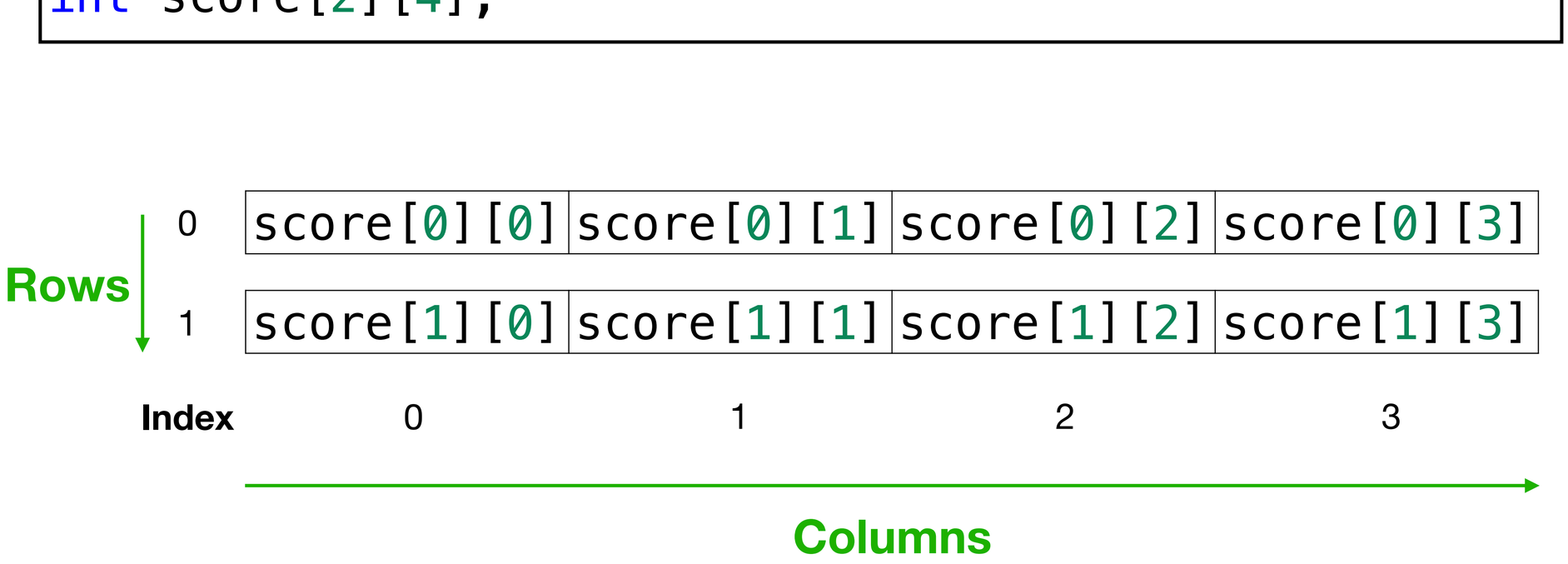
10/22 Recap for Arrays

W7 Slido: #3533560

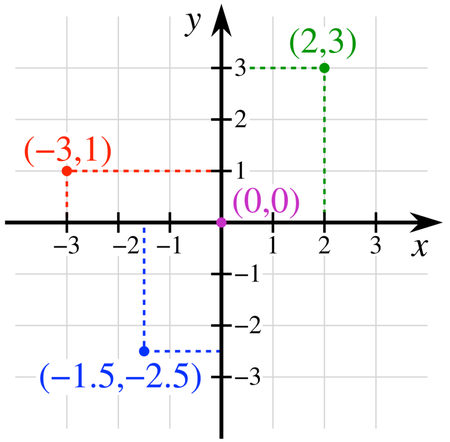
GitHub repo

What a 2-D Array is (in index)

```
int score[2][4];
```

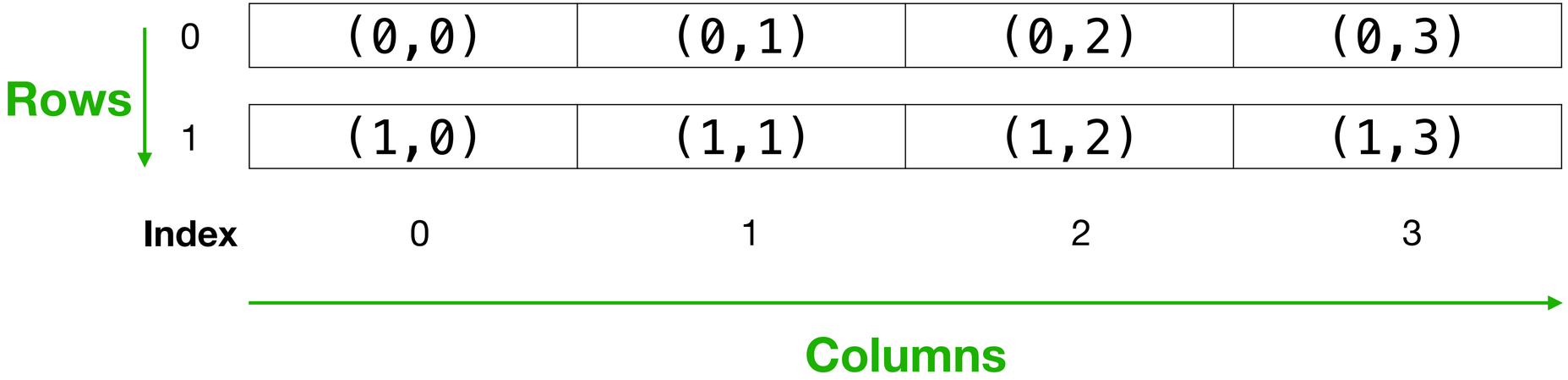


(Cartesian Coordinate System)

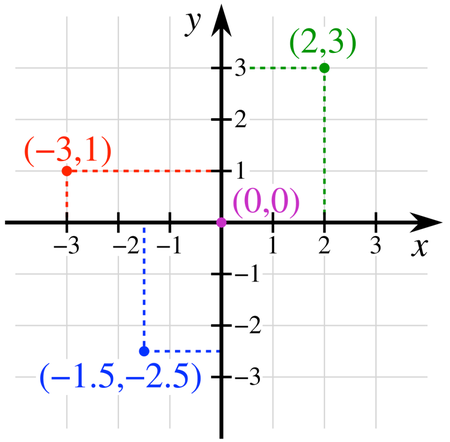


What a 2-D Array is (in coordinate)

```
int score[2][4];
```



(Cartesian Coordinate System)



W6 Quiz: The 4x4 spiral matrix

https://github.com/mcps5601/C-course-materials/blob/main/quizzes/w6_ans.c

```c
int N = 4;
int num = 1;
int arr[4][4] = {0};
int movements[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
int row = 0, col = 0;
int direction = 0;
```

(0,0)	(0,1)	(0,2)	(0,3)

(0,0)	(0,1)	(0,2)	(0,3)
			(1,3)
			(2,3)
			(3,3)

(0,0)	(0,1)	(0,2)	(0,3)
			(1,3)
			(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)			(1,3)
(2,0)			(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

W6 Quiz: The 4x4 spiral matrix

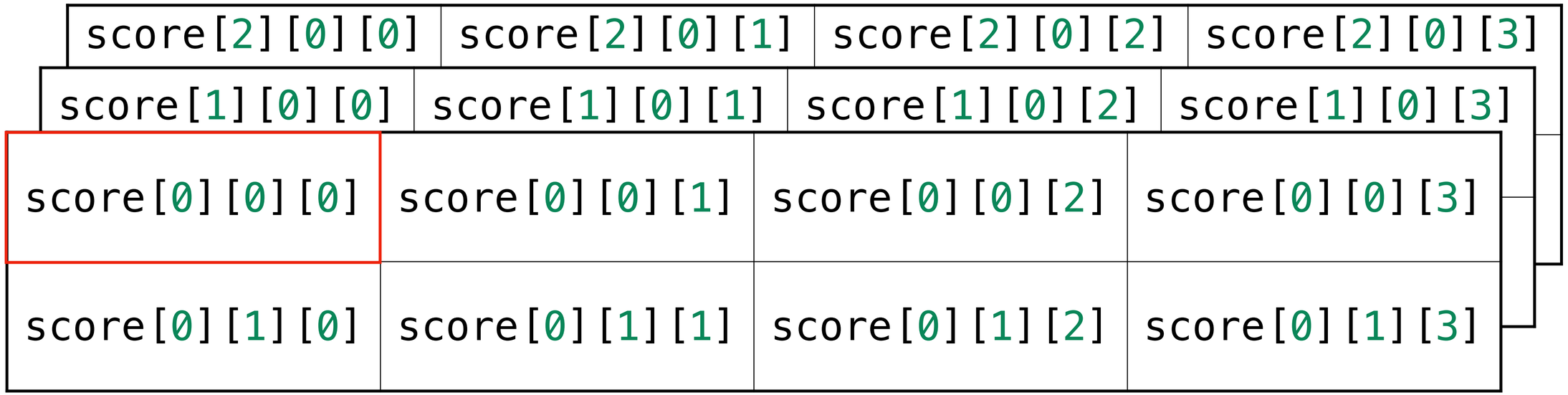
https://github.com/mcps5601/C-course-materials/blob/main/quizzes/w6_ans.c

```
while (num <= N * N){
    arr[row][col] = num++;

    int next_row = row + movements[direction][0];
    int next_col = col + movements[direction][1];

    // over the boundary or the next cell has been filled
    if (next_row < 0 || next_col < 0 || next_row >= N || next_col >= N ||
arr[next_row][next_col] != 0){
        direction = (direction + 1) % 4;
        next_row = row + movements[direction][0];
        next_col = col + movements[direction][1];
    }
    row = next_row;
    col = next_col;
}
```

Observation for the shape of a 3-D array

- Get the number of all elements of the array

```
int total_elements = sizeof(score) / sizeof(score[0][0][0]);
```



Visualization of a 3-D Array

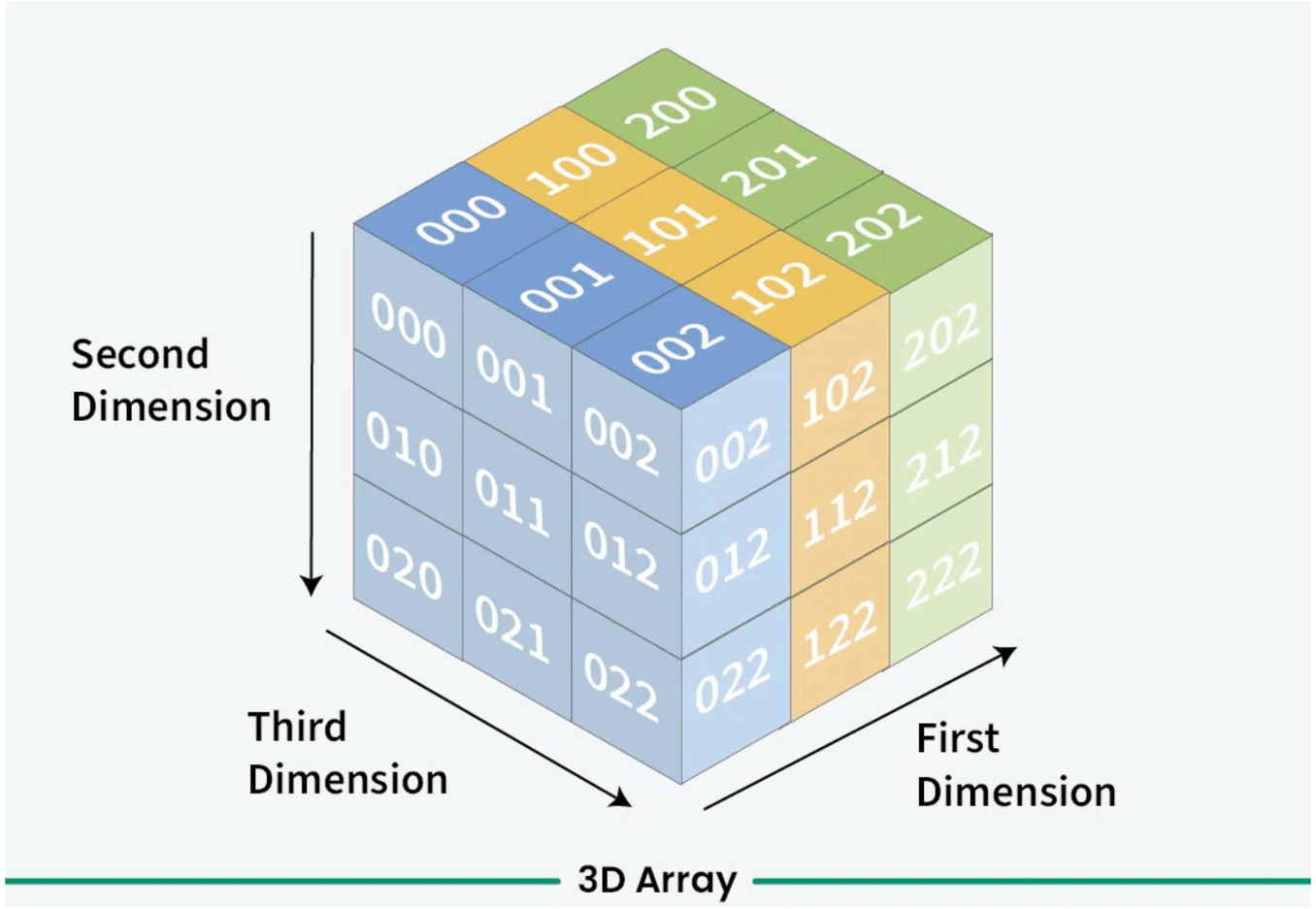


Figure source: https://www.geeksforgeeks.org/multidimensional-arrays-in-c/

Observation for a 3-D array

C-course-materials/04-Arrays/observe_3D.c

```c
int score[3][2][4] = {{
    {80, 85, 90, 100},
    {60, 65, 70, 100}
},
{
    {80, 85, 90, 100},
    {60, 65, 70, 100}
},
{
    {80, 85, 90, 100},
    {60, 65, 70, 100}
}};
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 4; j++) {
        printf("%2d ", score[2][i][j]);
    }
    printf("\n");
}
for (int i = 0; i < 4; i++) {
    printf("%2d ", score[0][1][i]);
}
```

Important Notes for Arrays

- In C, arrays are not treated as single objects that can be printed or returned at one time.
- Instead, we can get the elements according to **index**.

Memory Address of Arrays

scanf：用來進行輸入的函數

c_basics p.30

```
scanf(format, &變數1, &變數2, …);
```

- `scanf`： Read formatted data and store them according to the locations.
  - 可取得自鍵盤輸入的值
- format: 一段可包含 **format specifiers** 的字串 (string)
- &代表位置運算子，可以將數值存到變數的記憶體位置
- 同printf，第二個參數以後的數量與跟 **format specifiers** 一致

&: the Address Operator

- The address operator (&) returns the address of a variable (取址運算子).
- We can use the format specifier %p for printing an address.

```c
#include <stdio.h>
int main(){
    int score = 100;
    printf("The address of score: %p\n", &score);
}
```

Print the memory address of an array

C-course-materials/04-Arrays/print_array_addr.c

```c
#include <stdio.h>
int main(){
    int score[4] = {80, 85, 90, 100};
    printf("The address of this array: %p\n", &score);
    for (int i = 0; i < 4; i++){
        printf("The address of score[%d]: %p\n", i, &score[i]);
    }
}
```

Output

```
The address of this array: 0x7fffffffdd80
The address of score[0]: 0x7fffffffdd80
The address of score[1]: 0x7fffffffdd84
The address of score[2]: 0x7fffffffdd88
The address of score[3]: 0x7fffffffdd8c
```

The memory addresses are represented in base 16

二進制	十進制	十六進制
0000 0000	0	00
0000 0001	1	01
0000 0010	2	02
0000 0011	3	03
0000 0100	4	04
0000 0101	5	05
0000 0110	6	06
0000 0111	7	07
0000 1000	8	08
0000 1001	9	09
0000 1010	10	0A
0000 1011	11	0B
0000 1100	12	0C

```c
#include <stdio.h>
int main(){
    int num = 12;
    printf("%x", num);
}
```

- Use %x to print base 16 integers

Figure from: https://zh.wikipedia.org/zh-tw/ASCII

Properties of Array Address

- The first element of an array shares the address of the array.
- The memory addresses of an array in C are contiguous. For an int array:
  - arr[0] is at address A
  - arr[1] is at address A + 4
  - arr[2] is at address A + 8
  - arr[3] is at address A + 12

Memory Address for a Multi-dimensional Array

C-course-materials/04-Arrays/print_3Darray_addr.c

```c
#include <stdio.h>
int main(){
    int score[2][3][4] = {
        {
            {80, 85, 90, 100},
            {75, 80, 85, 90},
            {60, 65, 70, 75}
        },
        {
            {55, 60, 65, 70},
            {50, 55, 60, 65},
            {45, 50, 55, 60}
        }
    };
    printf("The address of score: %p\n", &score);
    printf("The address of score[0]: %p\n", &score[0]);
    printf("The address of score[0][0]: %p\n", &score[0][0]);
    printf("The address of score[0][0][0]: %p\n", &score[0][0][0]);
    printf("The address of score[1]: %p\n", &score[1]);
    printf("The address of score[1][0]: %p\n", &score[1][0]);
    printf("The address of score[1][0][0]: %p\n", &score[1][0][0]);
}
```

Memory Address for a Multi-dimensional Array

C-course-materials/04-Arrays/print_3Darray_addr.c

```c
#include <stdio.h>
int main(){
    int score[2][3][4] = {
        {
            {80, 85, 90, 100},
            {75, 80, 85, 90},
            {60, 65, 70, 75}
        },
        {
            {55, 60, 65, 70},
            {50, 55, 60, 65},
            {45, 50, 55, 60}
        }
    };
    printf("The address of this array: %p\n", &score);
    for (int i = 0; i < 2; i++){
        for (int j = 0; j < 3; j++){
            for (int k = 0; k < 4; k++){
                printf("The address of score[%d][%d][%d]: %p\n", i, j, k, &score[i][j][k]);
            }
        }
    }
}
```

C-course-materials/04-Arrays/print_3Darray_addr.c

```
The address of this array: 0x7fffffffdd20
The address of score[0][0][0]: 0x7fffffffdd20
The address of score[0][0][1]: 0x7fffffffdd24
The address of score[0][0][2]: 0x7fffffffdd28
The address of score[0][0][3]: 0x7fffffffdd2c
The address of score[0][1][0]: 0x7fffffffdd30
The address of score[0][1][1]: 0x7fffffffdd34
The address of score[0][1][2]: 0x7fffffffdd38
The address of score[0][1][3]: 0x7fffffffdd3c
The address of score[0][2][0]: 0x7fffffffdd40
The address of score[0][2][1]: 0x7fffffffdd44
The address of score[0][2][2]: 0x7fffffffdd48
The address of score[0][2][3]: 0x7fffffffdd4c
The address of score[1][0][0]: 0x7fffffffdd50
The address of score[1][0][1]: 0x7fffffffdd54
The address of score[1][0][2]: 0x7fffffffdd58
The address of score[1][0][3]: 0x7fffffffdd5c
The address of score[1][1][0]: 0x7fffffffdd60
The address of score[1][1][1]: 0x7fffffffdd64
The address of score[1][1][2]: 0x7fffffffdd68
The address of score[1][1][3]: 0x7fffffffdd6c
The address of score[1][2][0]: 0x7fffffffdd70
The address of score[1][2][1]: 0x7fffffffdd74
The address of score[1][2][2]: 0x7fffffffdd78
The address of score[1][2][3]: 0x7fffffffdd7c
```

Properties of Multi-dimensional Array Address

- **Contiguous Memory:**
  - The memory addresses of an array in C are contiguous.
  - Row major ordering
- **Same Address for Initial Elements**:
  - The address of score, score[0], score[0][0], and score[0][0][0] are the same, as they all point to the first element of the array.
  - The address of score[1], score[1][0], and score[1][0][0] are the same.