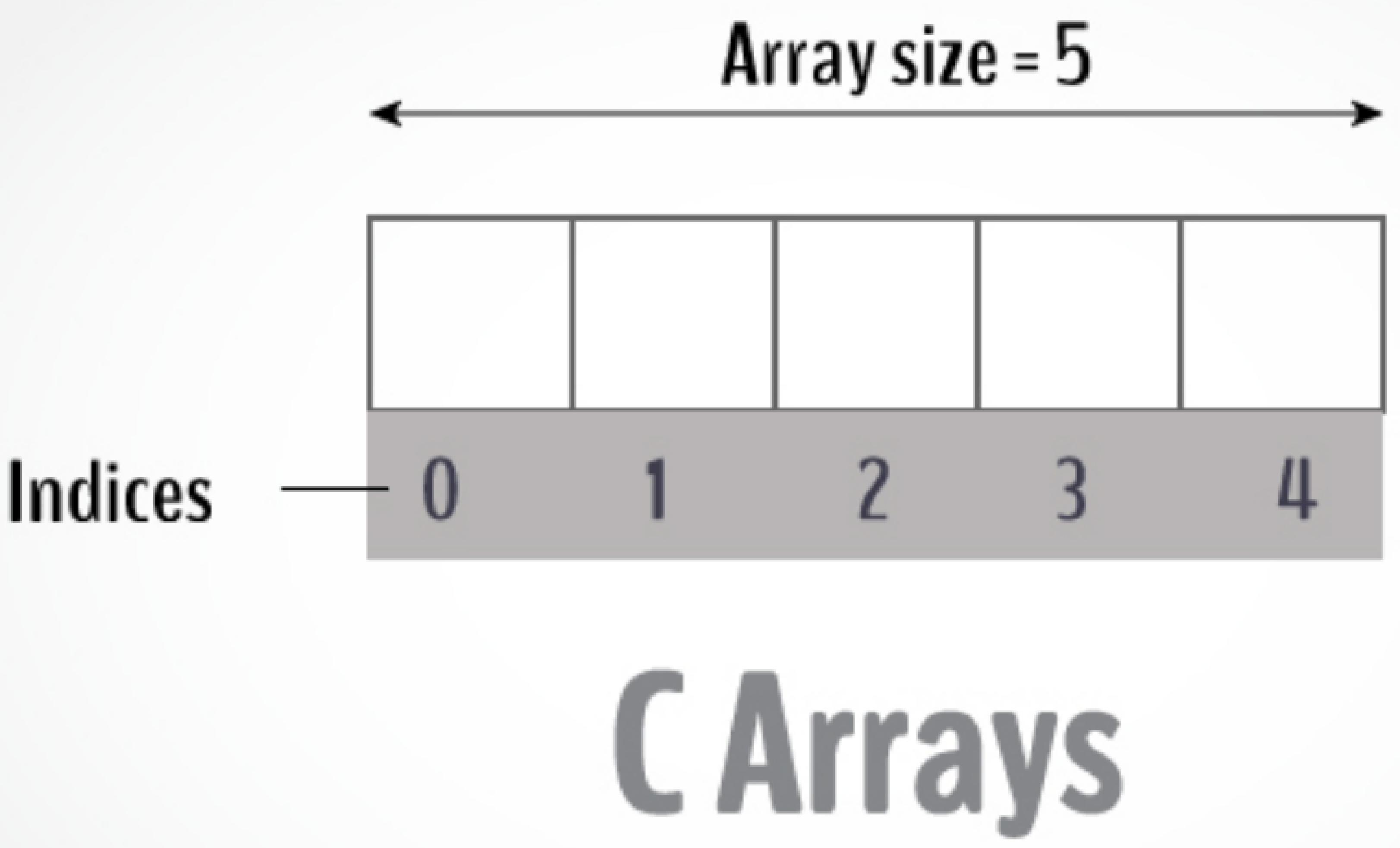
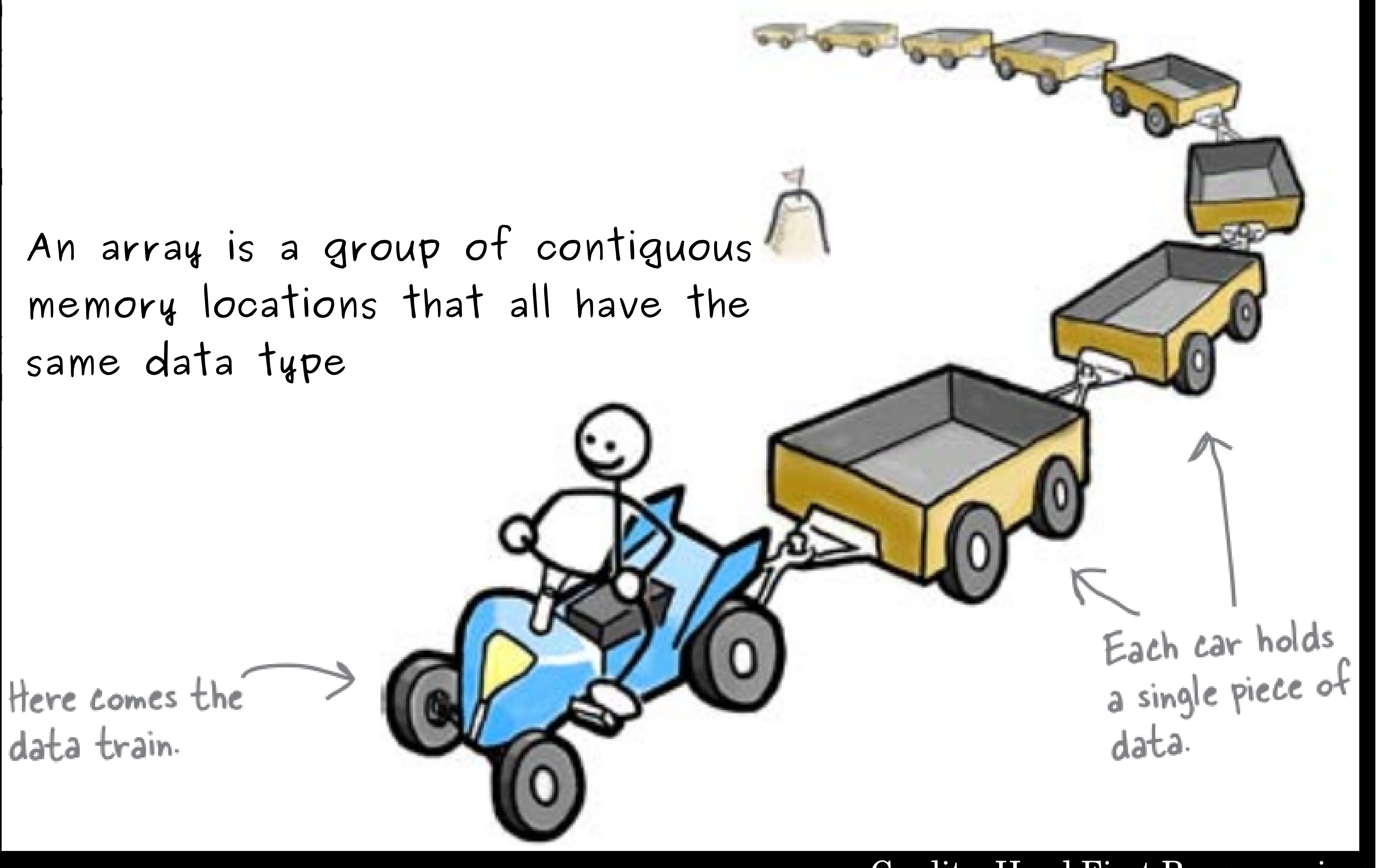
CSE102
Computer Programming



Credits: www.programiz.com

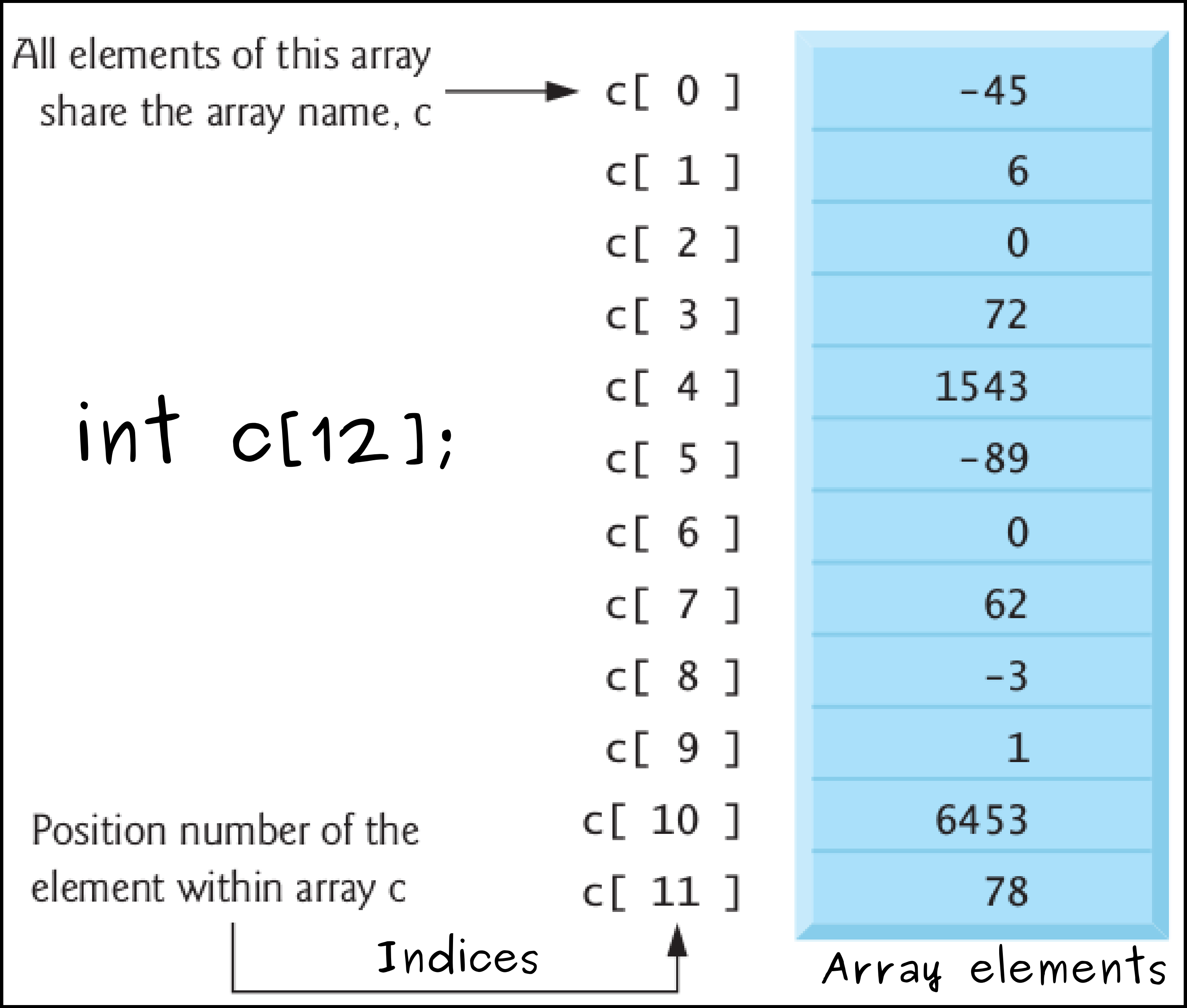
Array as a Data Train

An array is a group of contiguous memory locations that all have the same data type


Here comes the data train.
Each car holds a single piece of data.

Credits: Head First Programming

An Array



All elements of this array share the array name, c

```
int c[12];
```

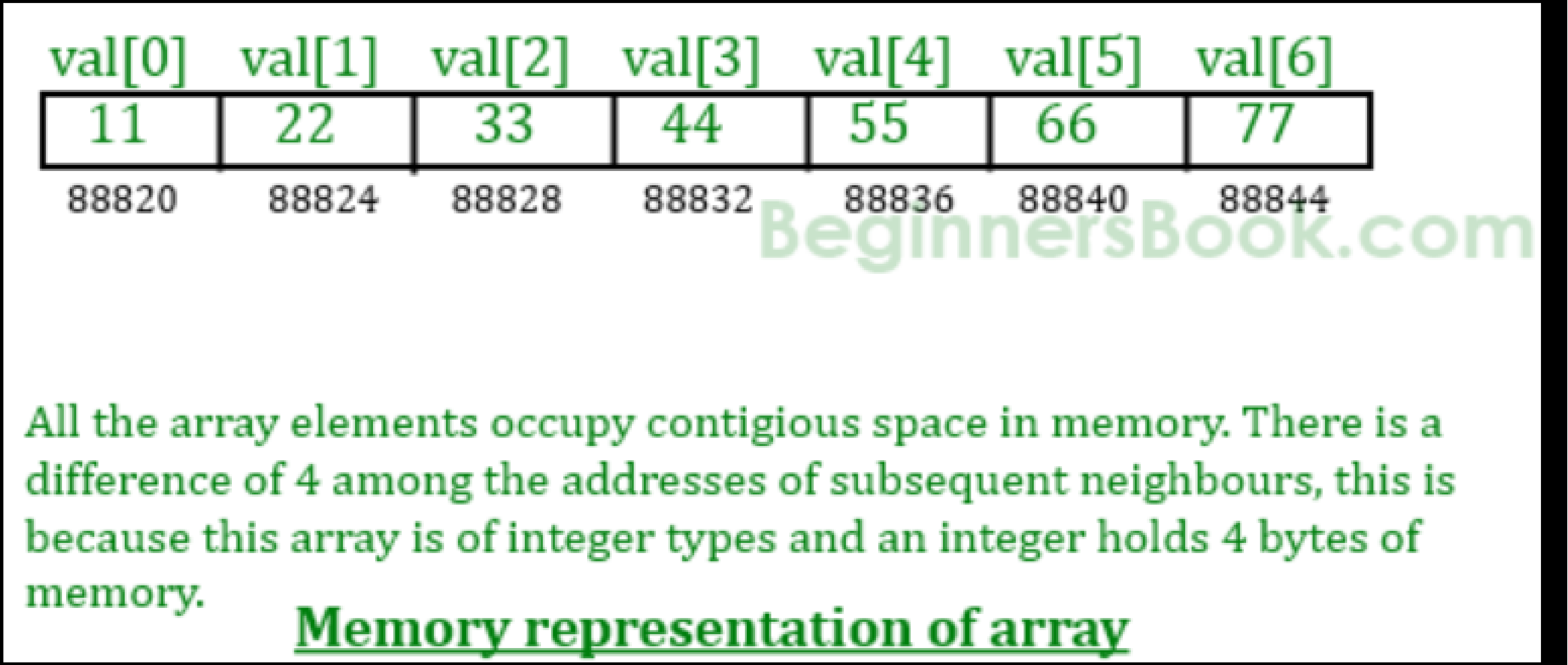
c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Position number of the element within array c

Indices

Array elements

Contiguous Memory



val[0]	val[1]	val[2]	val[3]	val[4]	val[5]	val[6]
11	22	33	44	55	66	77
88820	88824	88828	88832	88836	88840	88844

All the array elements occupy contigious space in memory. There is a difference of 4 among the addresses of subsequent neighbours, this is because this array is of integer types and an integer holds 4 bytes of memory.

Memory representation of array

Declaration

```
// Declare an integer array called marks
// with 66 elements

int marks[66];
// Use #define to specify size
#define SIZE 66
int marks[SIZE];
// Variable as array size
const int size = 66;
int marks[size];
```

Initialization

```
// Declare and initialize integer array
// with 3 elements
int nums[3] = {1,11,111};

// If length omitted compiler counts
// but only during init cum declaration
int nums[] = {1,11,111};
// Use {0} or {} to init all elements to 0
int nums[3] = {0};
int nums[3] = {};
```

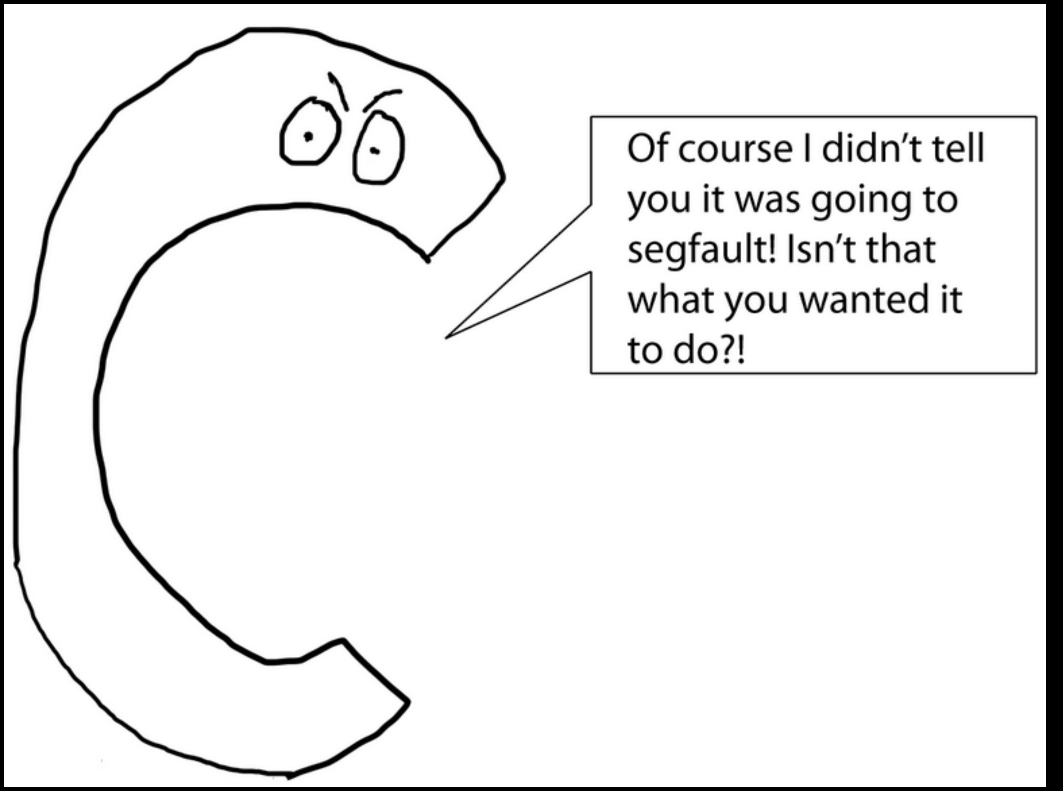
Initialization

```
// Number of elements in initialization
// shall be <= array size

// Remaining elements become 0 but
// confusing don't do this
int nums[3] = {1,11};
// Compiler Error too many initializers
int nums[3] = {1,11,111,1111};
```

Index-bound Check

```
const int size = 5;
int num[size]; // indices 0 to 4
// index out of bound
// still can be compiled!!! but …
num[88] = 555;
printf(“%d”,
       num[88]);
```


Of course I didn’t tell you it was going to segfault! Isn’t that what you wanted it to do?!

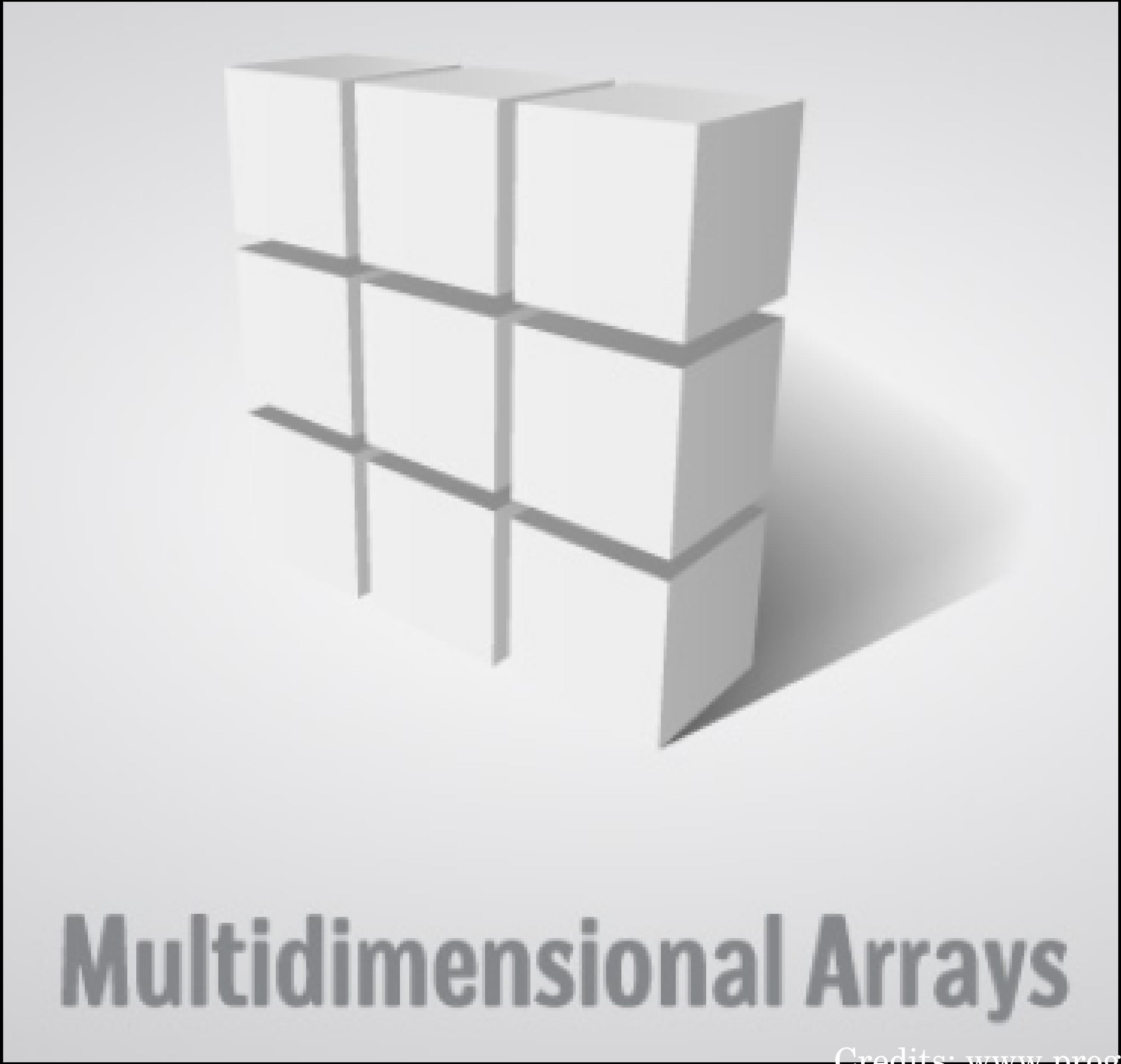
Credits: worldgnat.wordpress.com

Loops and Arrays

Arrays work hand-in-hand with loops. You can process all the elements of an array via a loop. For loop often turns out to be the most appropriate choice for processing arrays.

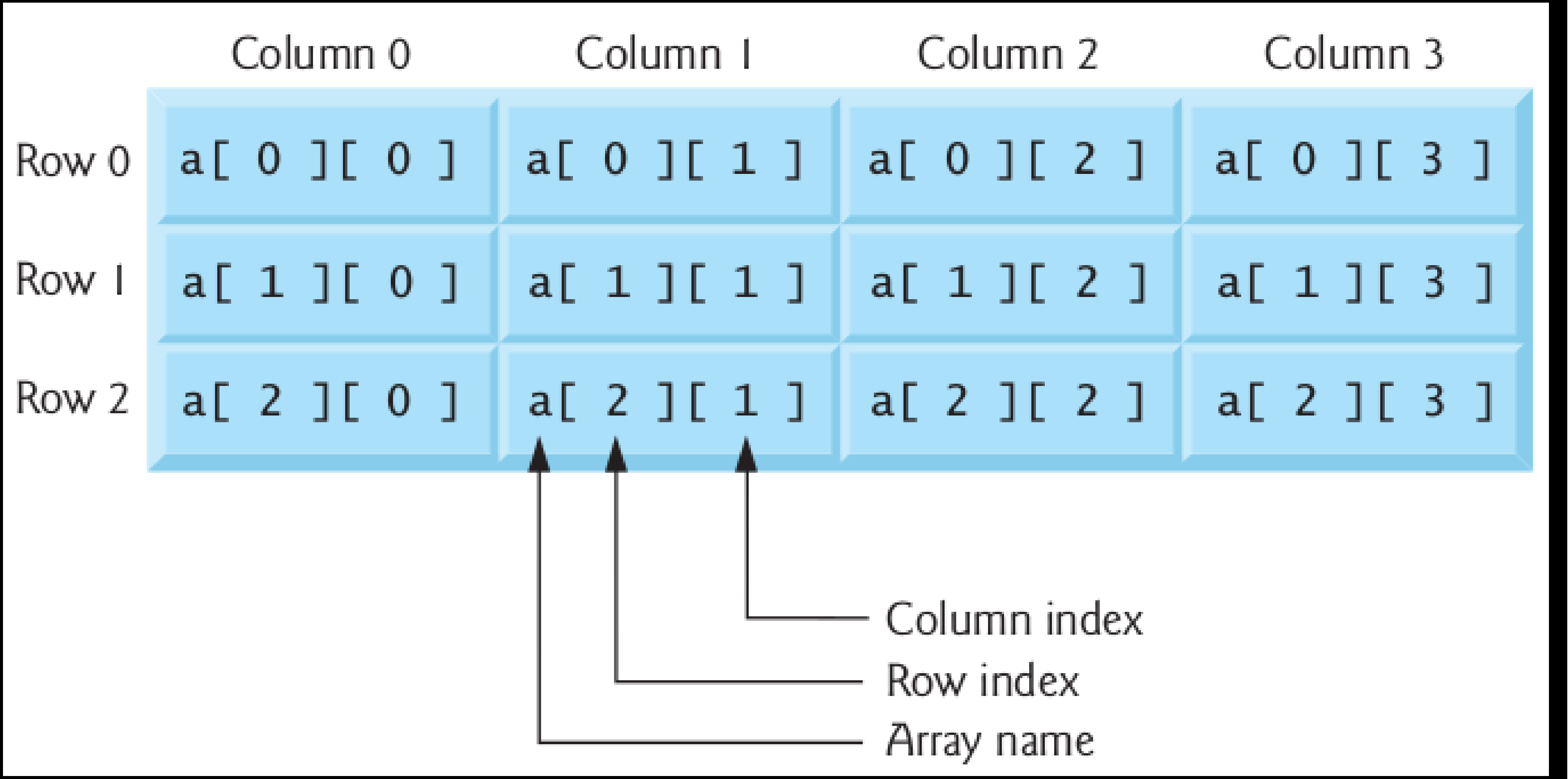
Functions and Arrays

You can also pass arrays into functions. but you need to pass the size of array too. Because there is no way to tell the size of array from the array argument inside the called function



Credits: www.programiz.com

2D Arrays



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

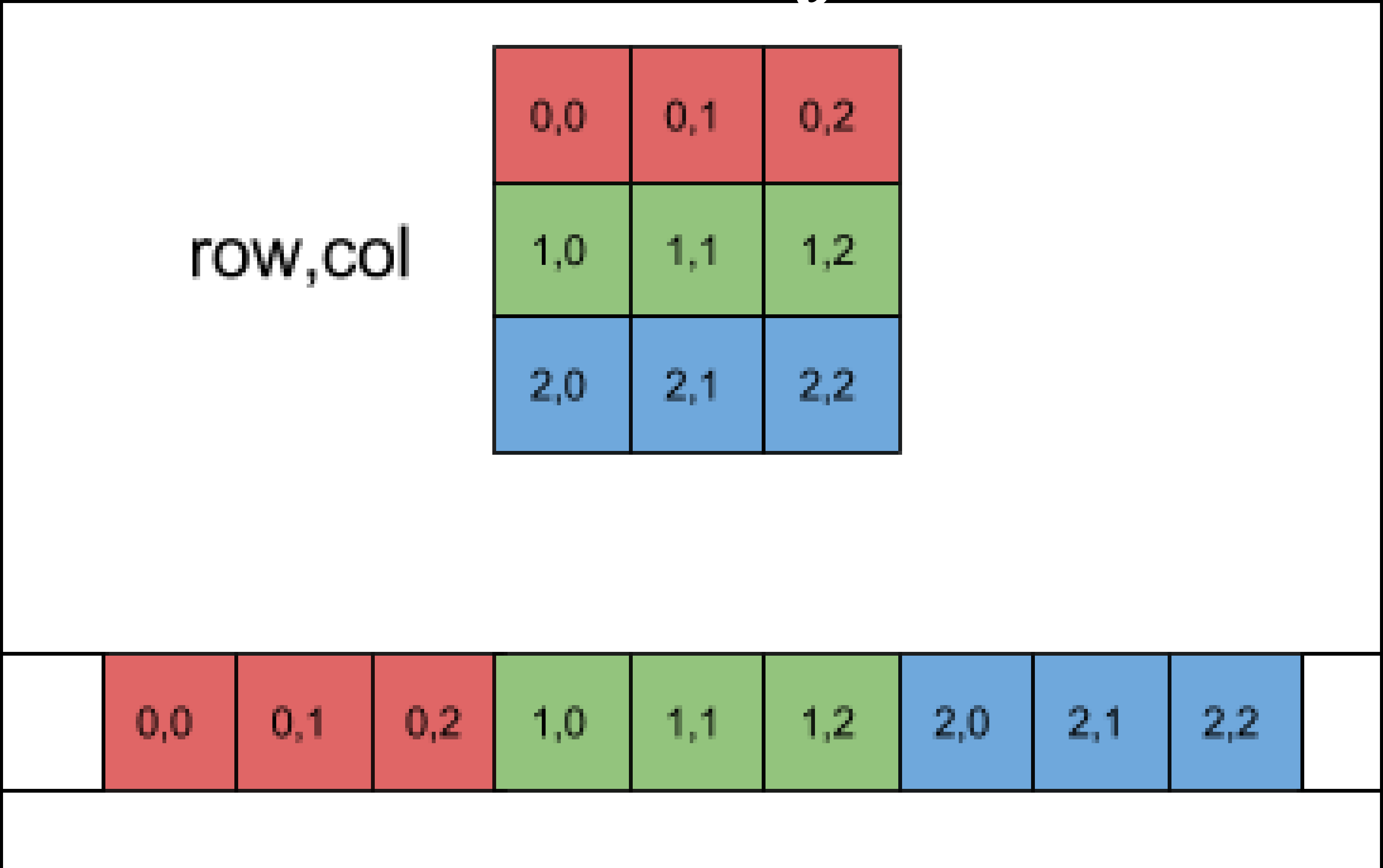
Credits: How to Program C

Initialization

```
// Declare and initialize 2D integer array
// with 3 elements in each row
int nums[][3] = {{1,11,111}
                 {2,22,222}};

// Observe that row index can be omitted
// and implied because C stores multi-
// dimensional arrays as row-major
```

Row-Major



row,col

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

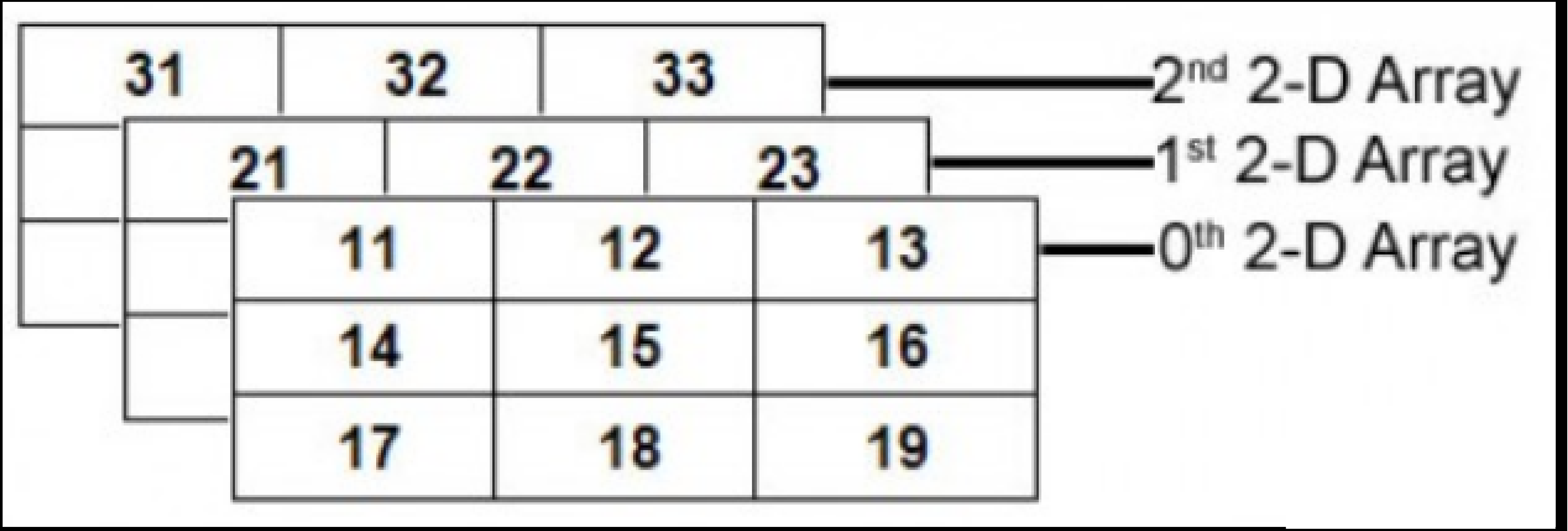
	0,0	0,1	0,2	1,0	1,1	1,2	2,0	2,1	2,2	

Thus given column size, row size can be implied as can be seen above

Credits: eli.thegreenplace.net

3D Arrays

```
int nums[3][3][3];
```



31	32	33

21	22	23

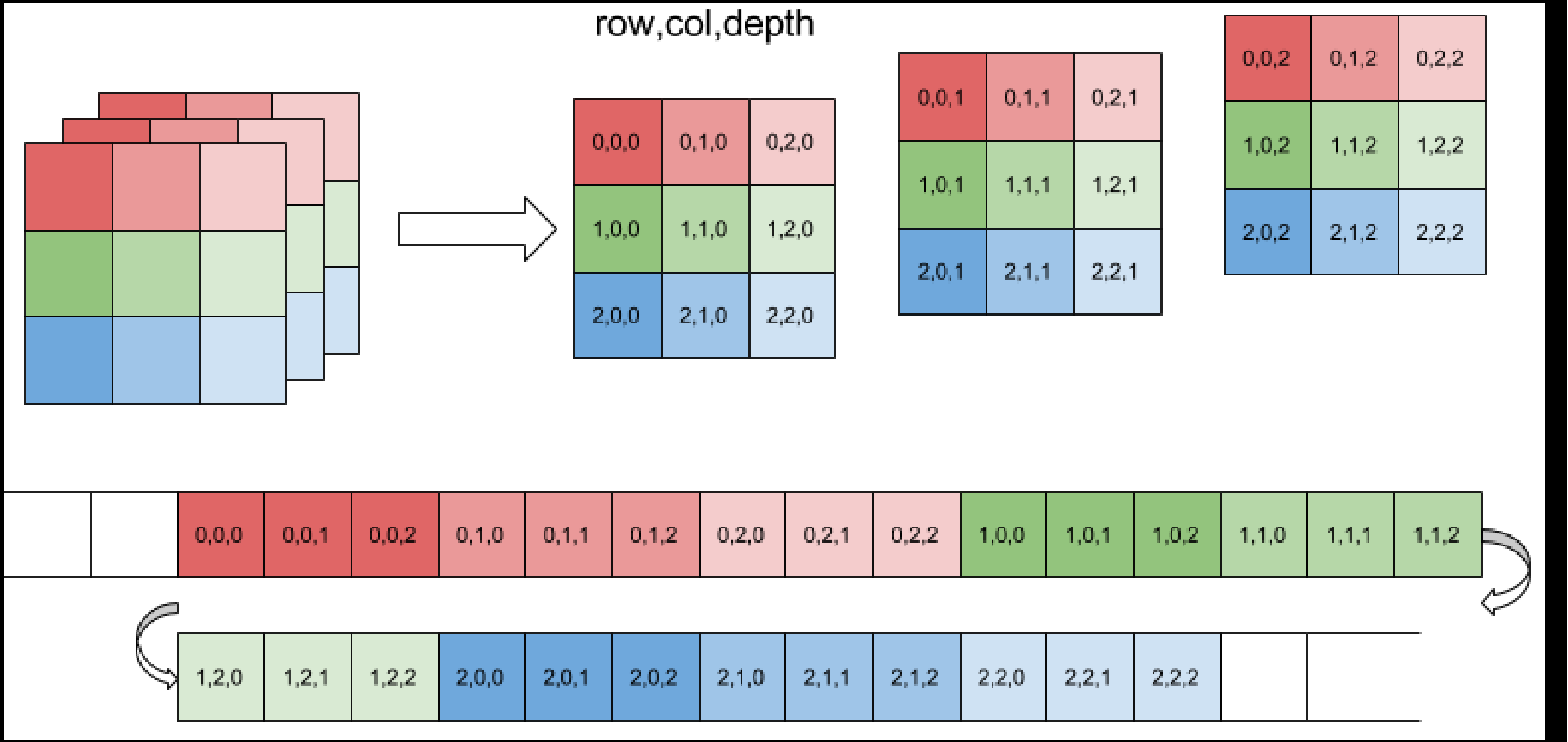
11	12	13
14	15	16
17	18	19

Credits: owlcation.com

Initialization

```
int nums[3][3][3]= {
            {
            {11, 12, 13},
            {14, 15, 16},
            {17, 18, 19}
            },
            {
            {21, 22, 23},
            {24, 25, 26},
            {27, 28, 29}
            },
            {
            {31, 32, 33},
            {34, 35, 36},
            {37, 38, 39}}, };
```

Row-Major



row,col,depth

0,0,0	0,1,0	0,2,0
1,0,0	1,1,0	1,2,0
2,0,0	2,1,0	2,2,0

0,0,1	0,1,1	0,2,1
1,0,1	1,1,1	1,2,1
2,0,1	2,1,1	2,2,1

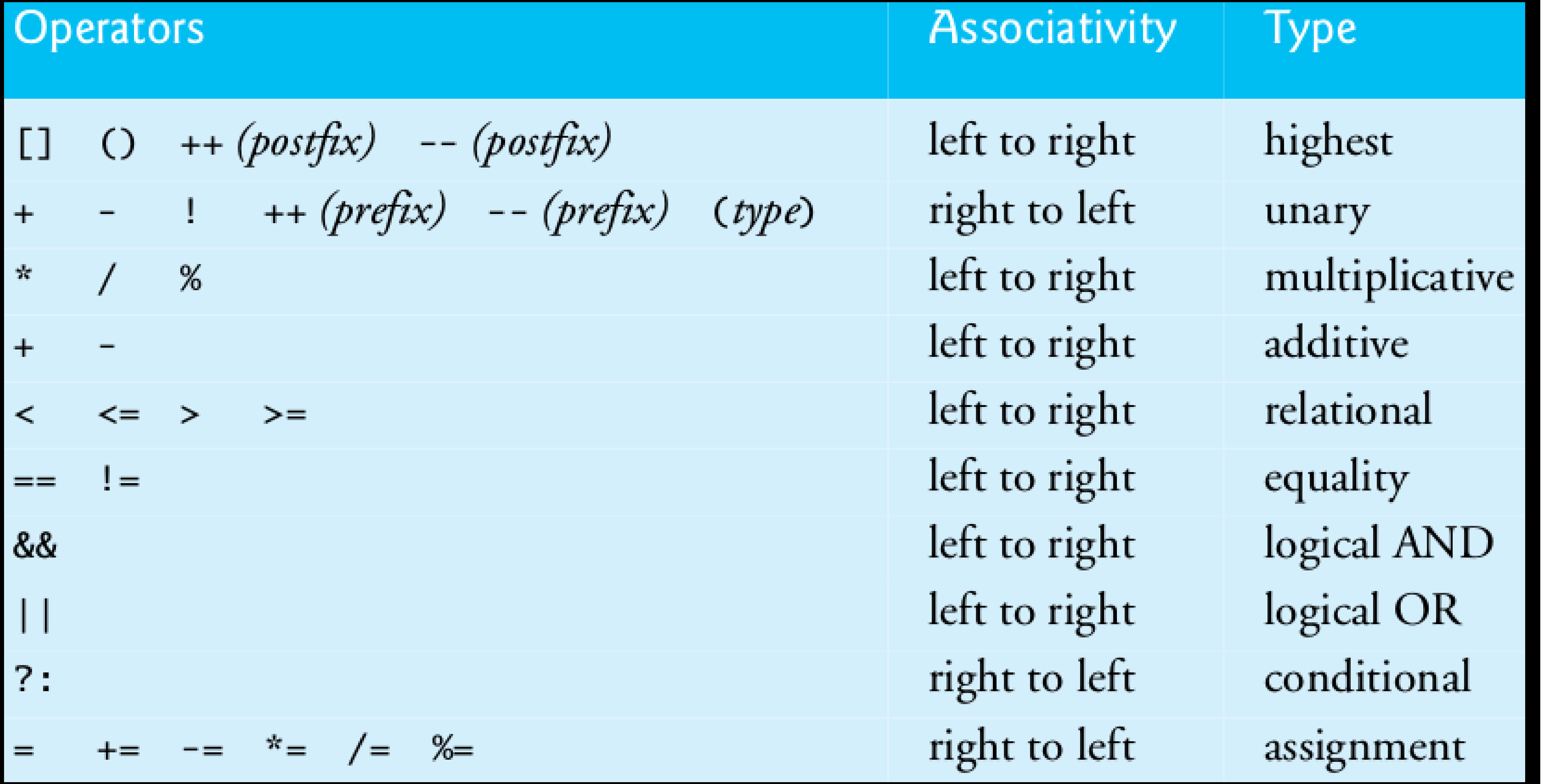
0,0,2	0,1,2	0,2,2
1,0,2	1,1,2	1,2,2
2,0,2	2,1,2	2,2,2

		0,0,0	0,0,1	0,0,2	0,1,0	0,1,1	0,1,2	0,2,0	0,2,1	0,2,2	1,0,0	1,0,1	1,0,2	1,1,0	1,1,1	1,1,2

1,2,0	1,2,1	1,2,2	2,0,0	2,0,1	2,0,2	2,1,0	2,1,1	2,1,2	2,2,0	2,2,1	2,2,2		

Credits: eli.thegreenplace.net

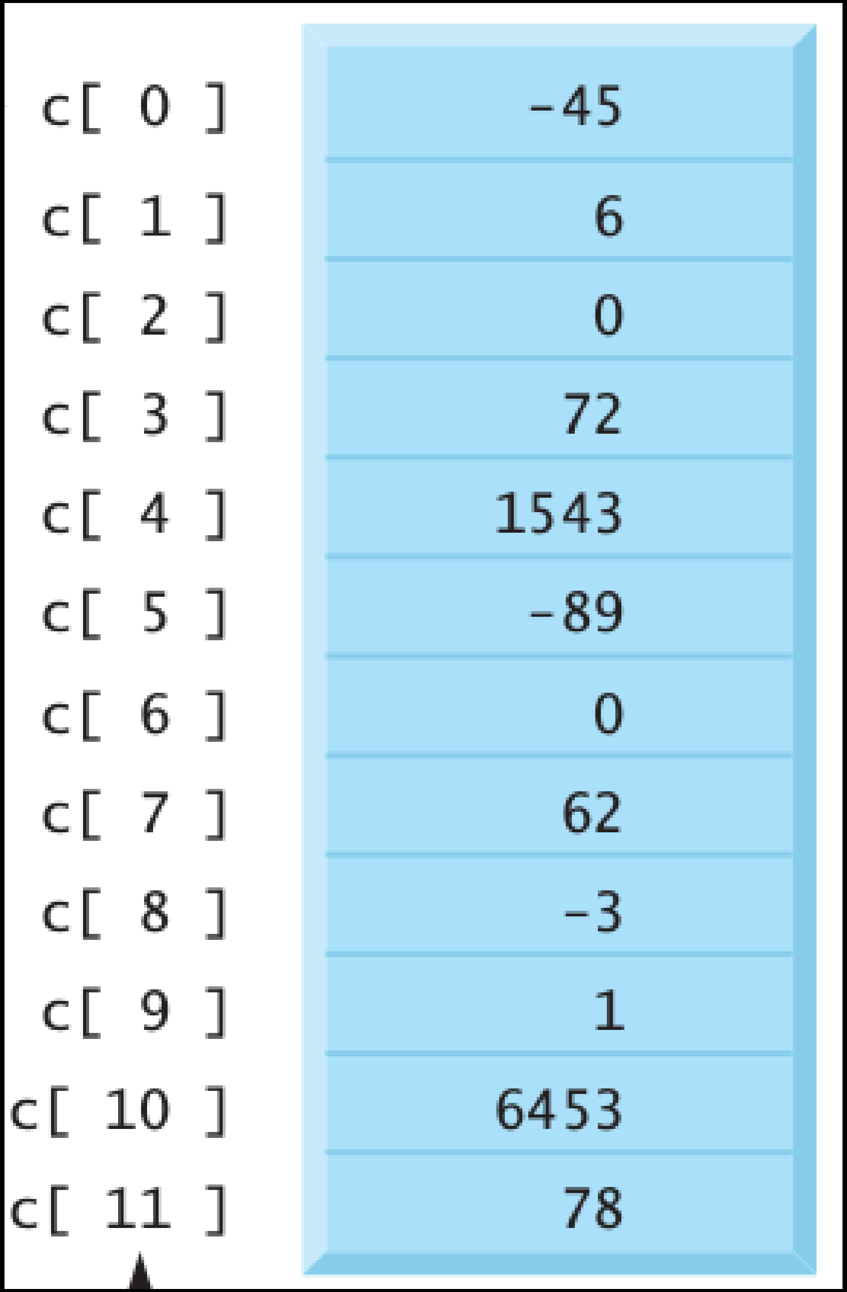
[ ] is an Operator too!!



Operators	Associativity	Type
[] () ++ *(postfix)* -- *(postfix)*	left to right	highest
+ - ! ++ *(prefix)* -- *(prefix)* (*type*)	right to left	unary
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

Credits: How to Program C

Index can be Expressions!!



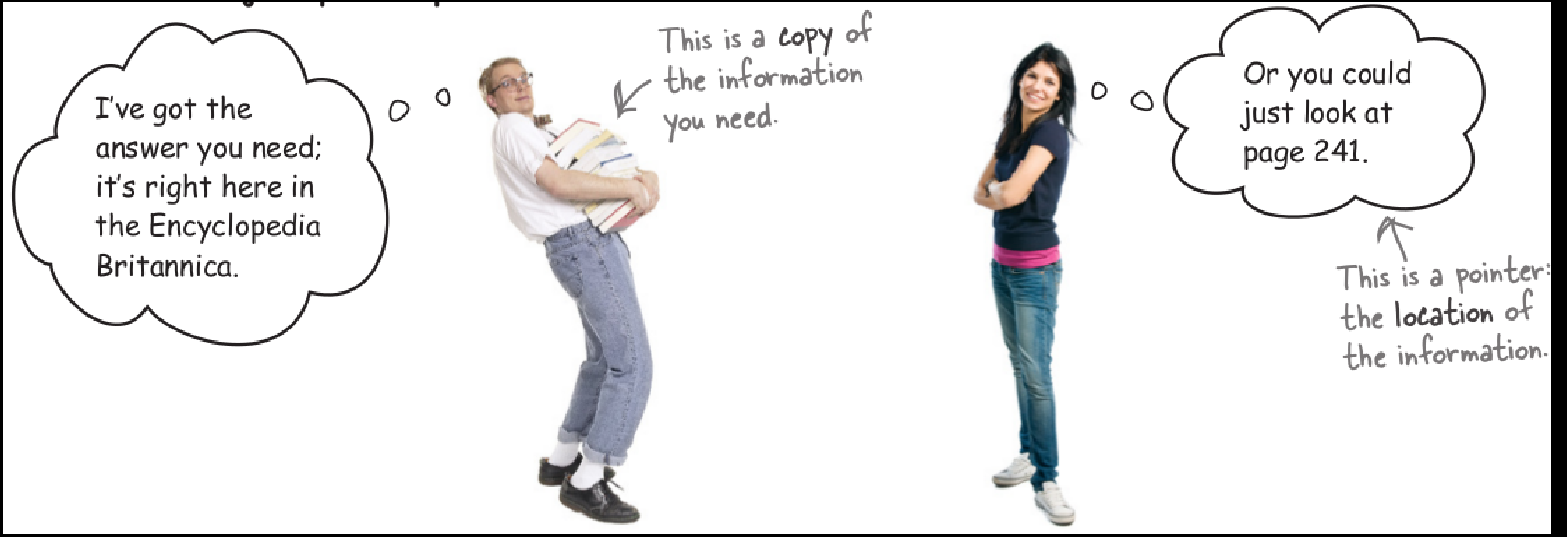
c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

```
// If a=5 and b=6
// then
c[a+b] += 2;
// changes c[11]
// to 80!!
```

Credits: How to Program C

CSE102
Computer Programming

(Next Topic)



I've got the answer you need; it's right here in the Encyclopedia Britannica.
This is a copy of the information you need.
Or you could just look at page 241.
This is a pointer: the location of the information.

Credits: Head First C