Arrays

CS 1044

Introduction to Arrays

- Recall: A variable is a slot that holds a single value
- Arrays are **collections** of values – multiple slots that each hold a value
  - They are **ordered** – there is a first value, a second value, and so on
  - They are **homogeneous** – every slot is the same type

Definitions

- The individual values in the array are called its **elements**
- The number of elements is the **length** of the array
- Each element is identified by its position in the array, called its **index**
  - Indices start at **0** and go up to ***length* – 1**

Declaring Array Variables

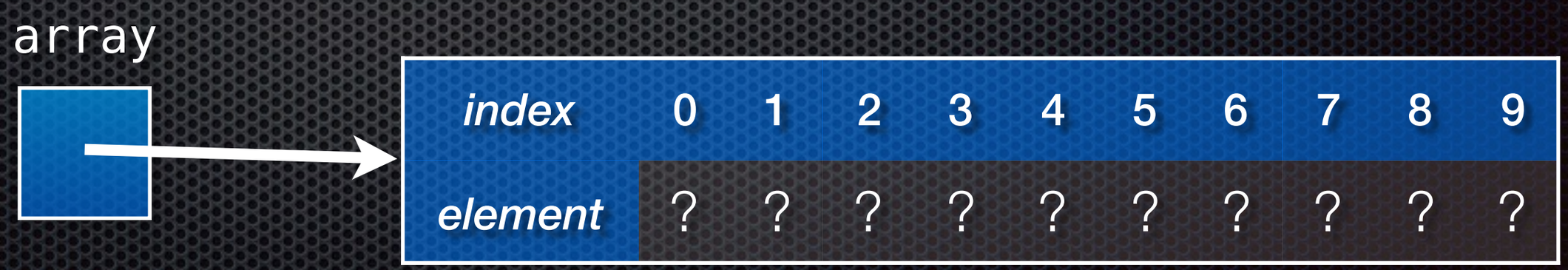
```
type name[length];
```

- ***type***: The type of each element in the array
- ***length***: The number of elements in the array (constant)
- **Notice:** it looks just like a regular variable declaration, but with the number of elements after the name

Declaring Array Variables

```
int array[10];
```

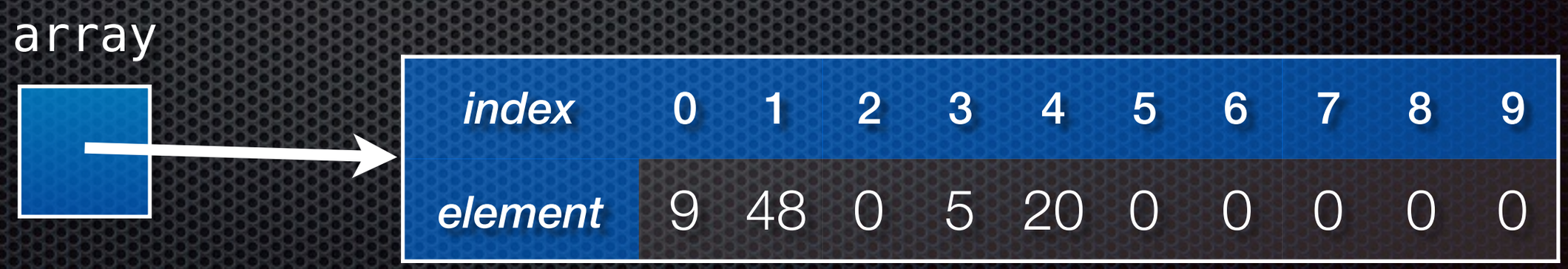
- Visualize an array as a row of boxes, with an index and element
- Like regular variables, elements of an array have **garbage values** by default



Initializing Array Variables

```
int array[10] = { 9, 48, 0, 5, 20 };
```

- Shortcut to initialize an array with values
- Values are initialized in order from index 0 – leftover slots get default values (0 for numbers, empty string for strings)



Accessing Elements

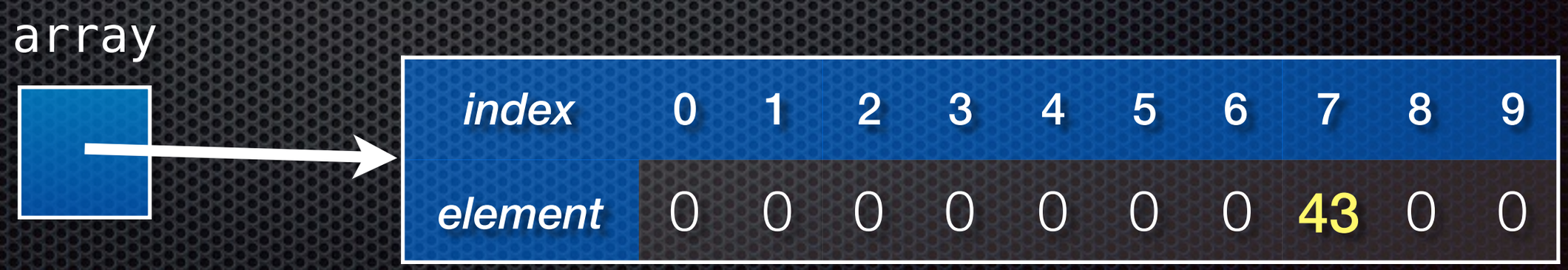
- Getting an element:

```
int x = array[5];
```

[ ] is called **"subscripting"** – read this aloud as **"array sub 5"**

- Setting an element:

```
array[7] = 43;
```



More about Array Length

- Array length must a **constant** known at **compile-time** (in other words, written into your source code)
- This won't work:

```
int length;
cout << "How many elements? ";
cin >> length;

int array[length];
```

Compiler error here

More about Array Length

- **Arrays don't remember their length** – given the name of an array, there's no way to figure out how long it is
- If you pass an array to a function, you **have to pass its length** as well, as a **separate parameter**

Nope

```
int sum(int array[])
{
  for (int i = 0; i < ???; i++)
}
```

Yup

```
int sum(int array[], int len)
{
  for (int i = 0; i < len; i++)
}
```

Array Indexing

- It may be strange at first to understand that array indices start at **0** instead of 1
- **Deal with it**

- Starting at zero happens **everywhere** in computer science, and other places too, so don't try to work around it

Loops and Arrays

- We frequently use `for` loops with arrays to loop over the elements and do something with each one

```
int array[N];

for (int i = 0; i < N; i++)
{
    // Do something with array[i]
}
```

2D Arrays

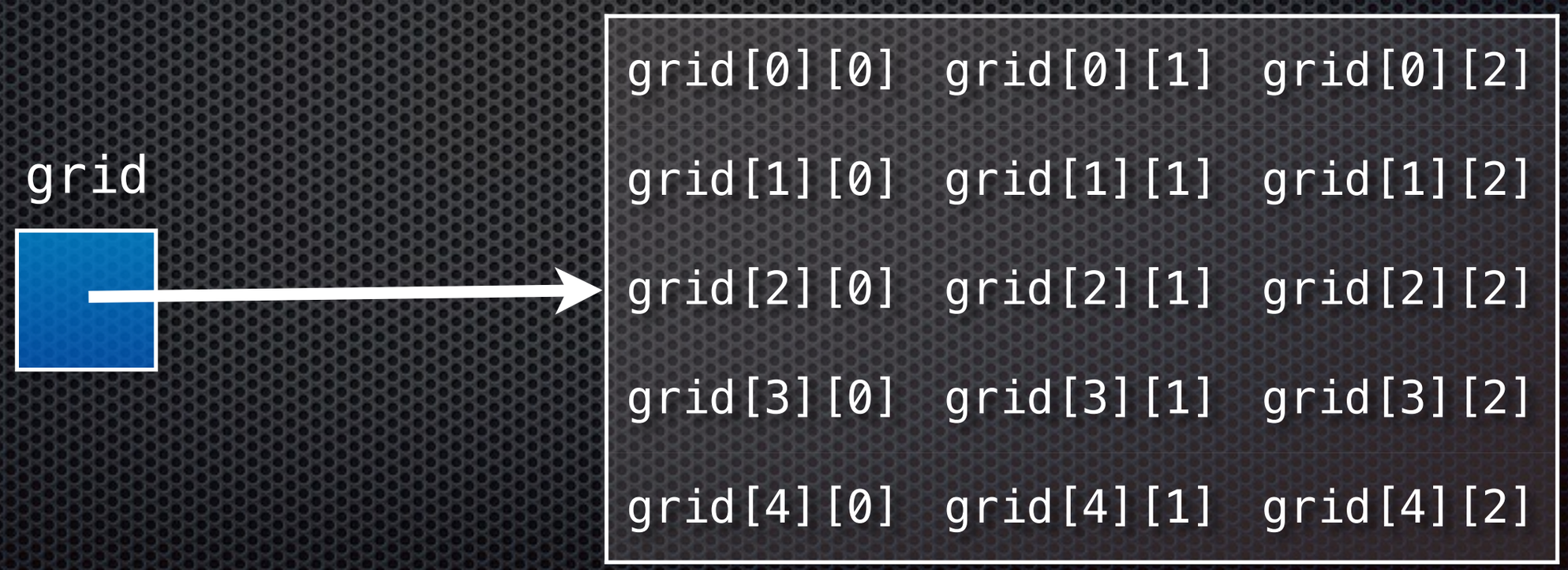
```
type name[rows][columns];
```

- ***type***: The type of each element in the grid/matrix
- ***rows***: The number of rows in the grid/matrix
- ***columns***: The number of columns in the grid/matrix

2D Arrays

```
int grid[5][3];
```

- Elements are accessed using the row index first, followed by the column index



Loops and 2D Arrays

- We can use two nested `for` loops to loop over all of the elements in a 2D array

```
int array[R][C];

for (int row = 0; row < R; row++)
{
    for (int col = 0; col < C; col++)
    {
        // Do something with grid[row][col]
    }
}
```

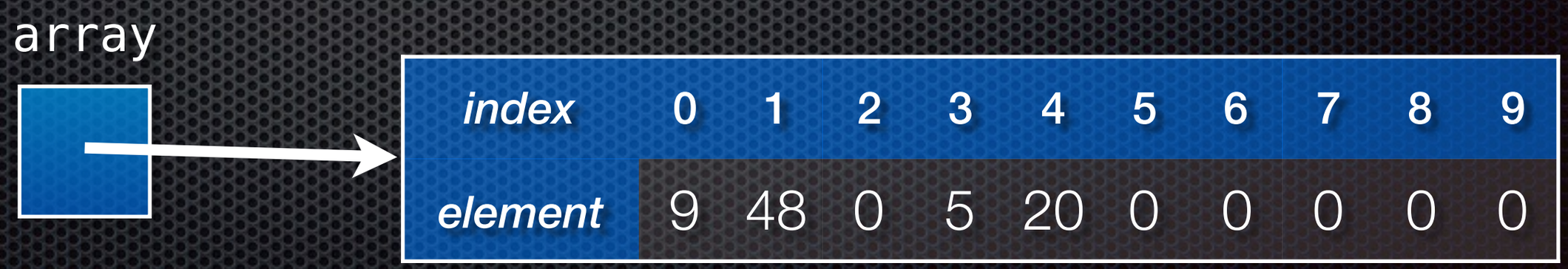
Array Drawbacks

- Length must be predetermined constant – can't depend on user input or other computations
- Length can never change
- Passing arrays to functions is inconvenient – must also pass the length separately
- Cannot easily insert or remove from the middle of an array
- Cannot return an array from a function

Array Extended Example

```
int array[10] = { 9, 48, 0, 5, 20 };
```

- Lets use this example from earlier in the slides
- Like we saw before this array will have 10 slots holding **int** variables, 5 have values the rest will be 0



Loops and Array Errors

- Using the example from the last slide
- We have an array with **10 elements**, conveniently called ‘array’
- The array has indices from **0 - 9**

```
// Lets increment all of the array values.
for (int i = 0; i < 10; i++)
{
        array[i]++;
}
```

Loops and Array Errors

```
// i starts out as 0.
for (int i = 0; i < 10; i++)
{
        // In this example we have
        // just completed i = 2.
        array[i]++;
}
```

array

index	0	1	2	3	4	5	6	7	8	9
element	9	48	0	5	20	0	0	0	0	0

array

index	0	1	2	3	4	5	6	7	8	9
element	10	49	1	5	20	0	0	0	0	0

Loops and Array Errors

```
// After all of the array elements
// have been incremented.
for (int i = 0; i < 10; i++)
{
        // Increments elements 0 – 9,
        // just completed index 9.
        array[i]++;
}
```

array

index	0	1	2	3	4	5	6	7	8	9
element	9	48	0	5	20	0	0	0	0	0

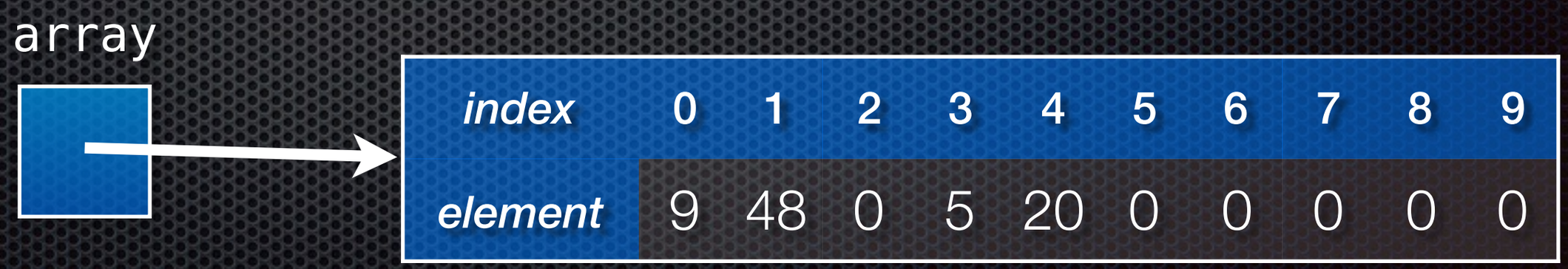
array

index	0	1	2	3	4	5	6	7	8	9
element	10	49	1	6	21	1	1	1	1	1

Array Errors

```
int array[10] = { 9, 48, 0, 5, 20 };
```

- Lets reuse this example from earlier in the slides.
- Like we saw before this array will have 10 slots holding **int** variables, 5 have values the rest will be 0



Loops and Array Errors

- Using the example from the last slide
- We have an array with **10 elements**, conveniently called ‘array’
- The array has indices from **0 - 9**

```
// This is an easy error to make.
for (int i = 0; i <= 10; i++)
{
        array[i]++;
}
```

Loops and Array Errors

```
// Everything starts out according to plan.
for (int i = 0; i <= 10; i++)
{
        // Like last time, we have
        // just completed i = 2.
        array[i]++;
}
```

array

index	0	1	2	3	4	5	6	7	8	9
element	9	48	0	5	20	0	0	0	0	0

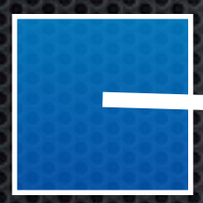
array

index	0	1	2	3	4	5	6	7	8	9
element	10	49	1	5	20	0	0	0	0	0

Loops and Array Errors

```
// That is until we get to the end.
for (int i = 0; i <= 10; i++)
{
        // Even though the array only has
        // elements 0 - 9 this loop lets
        // us access element 10, not good!
        array[i]++;
}
```

array



index	0	1	2	3	4	5	6	7	8	9
element	9	48	0	5	20	0	0	0	0	0

array

index	0	1	2	3	4	5	6	7	8	9
element	10	49	1	6	21	1	1	1	1	1

?

Frequent Array Errors

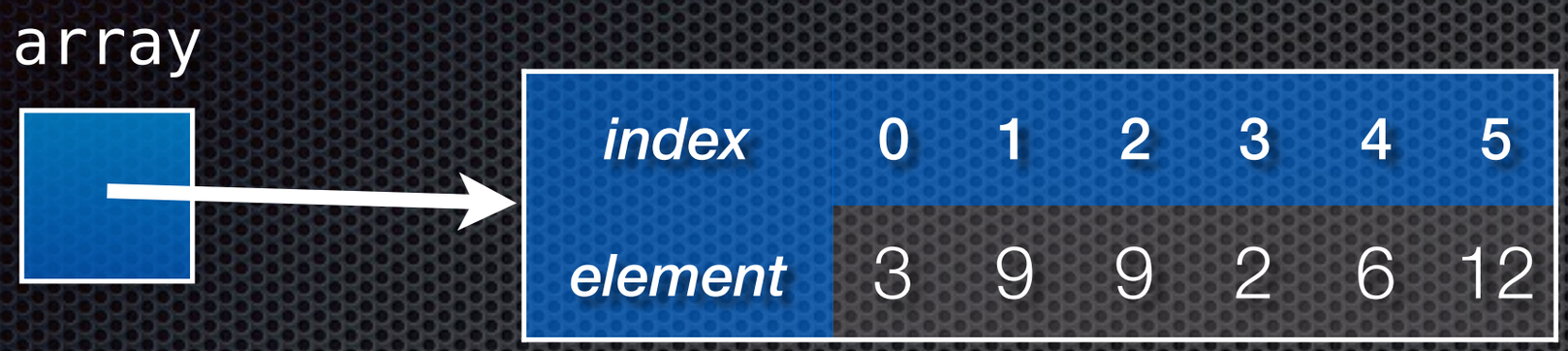
- Remember: the index you use to access an array element must be between **0** and ***length* – 1**
- If you use an index outside this range, your program might do one of three things:
  - Crash
  - Not crash, but give wrong results
  - Not crash and give correct results, sometimes

Advanced Array Example

- We can use arrays and functions to do something less trivial:

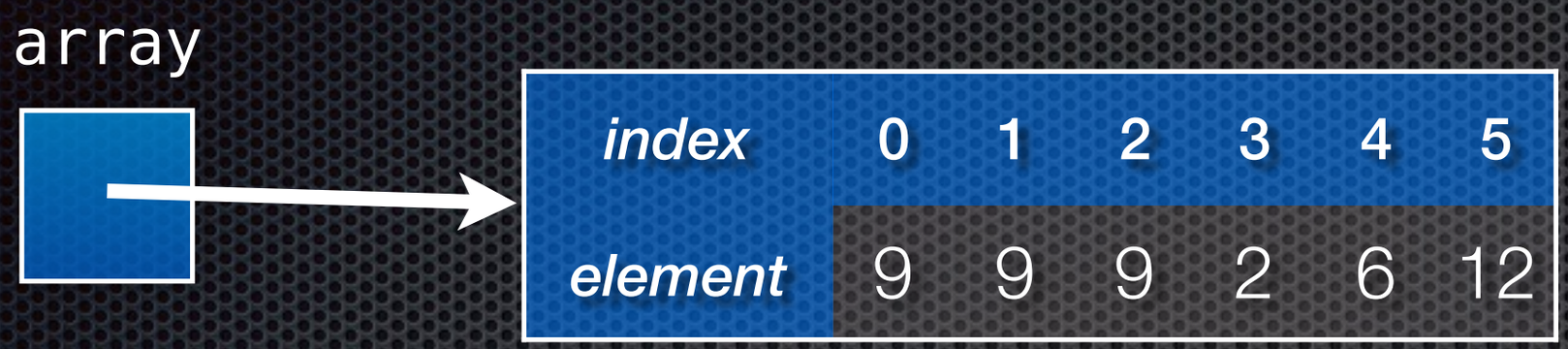
```
// Preconditions:
//     ‘A[]’ contains ‘AUsage’ values
// Post:
//   Barring an error, each element of
//   A[] has been replaced by its
//   immediate successor, IF it divided
//   its successor evenly (i.e., without
//   a remainder).

void advanced_array(int A[], int AUsage);
```



So we want to divide 9 by 3 and if there is no remainder copy the value 9 at index 1 to index 0

We can use ’%” or “mod” to get the remainder
9 % 3 is 0, meaning no remainder



So now we move on to the next index 1, its successor (index 2) has a value of 9

9 % 9 is 0, so we copy 9 at index 2 into index 1

array

index	0	1	2	3	4	5
element	3	9	9	2	6	12

array

index	0	1	2	3	4	5
element	9	9	9	6	12	12

Here is the final result

```
void advanced_array(int A[], int AUsage)
{
    // Array traversal, EXCEPT that you must avoid
    // looking for a successor to the last element
    // in the array:
    for (int idx = 0; idx < AUsage - 1; idx++)
    {

        // Subtlety:  you have to avoid dividing by zero:
        if ((A[idx] != 0) && (A[idx + 1] % A[idx] == 0))
        {
            A[idx] = A[idx + 1];
        }
    }
}
```