C++

Arrays in C++

Dr. Fouad A. Yaseen

Electronics & Communication Department

Lecture # 10

Arrays

String is a collection of characters. There are two types of strings commonly used in C++ programming language:

❑ Strings that are objects of string class (The Standard C++ Library string class)

❑ C-strings (C-style Strings)

C-strings

In C programming, the collection of characters is stored in the form of arrays. This is also supported in C++ programming. Hence it's called C-strings.

C-strings are arrays of type char terminated with null character, that is, \0 (ASCII value of null character is 0).

Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by **square brackets and specify the number of elements** it should store:

```
string cars[4];
```

We have now declared a variable that holds an array of four strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
```

Arrays

```
string cars[4] = {"Volvo", "BMW", "Ford", "Mazda"};
```

Access the Elements of an Array. You access an array element by referring to the index number. This statement accesses the value of the **first element** in **cars**:

To create an array of three integers, you could write:

```
int myNum[3] = {10, 20, 30};
```

Arrays

Example:

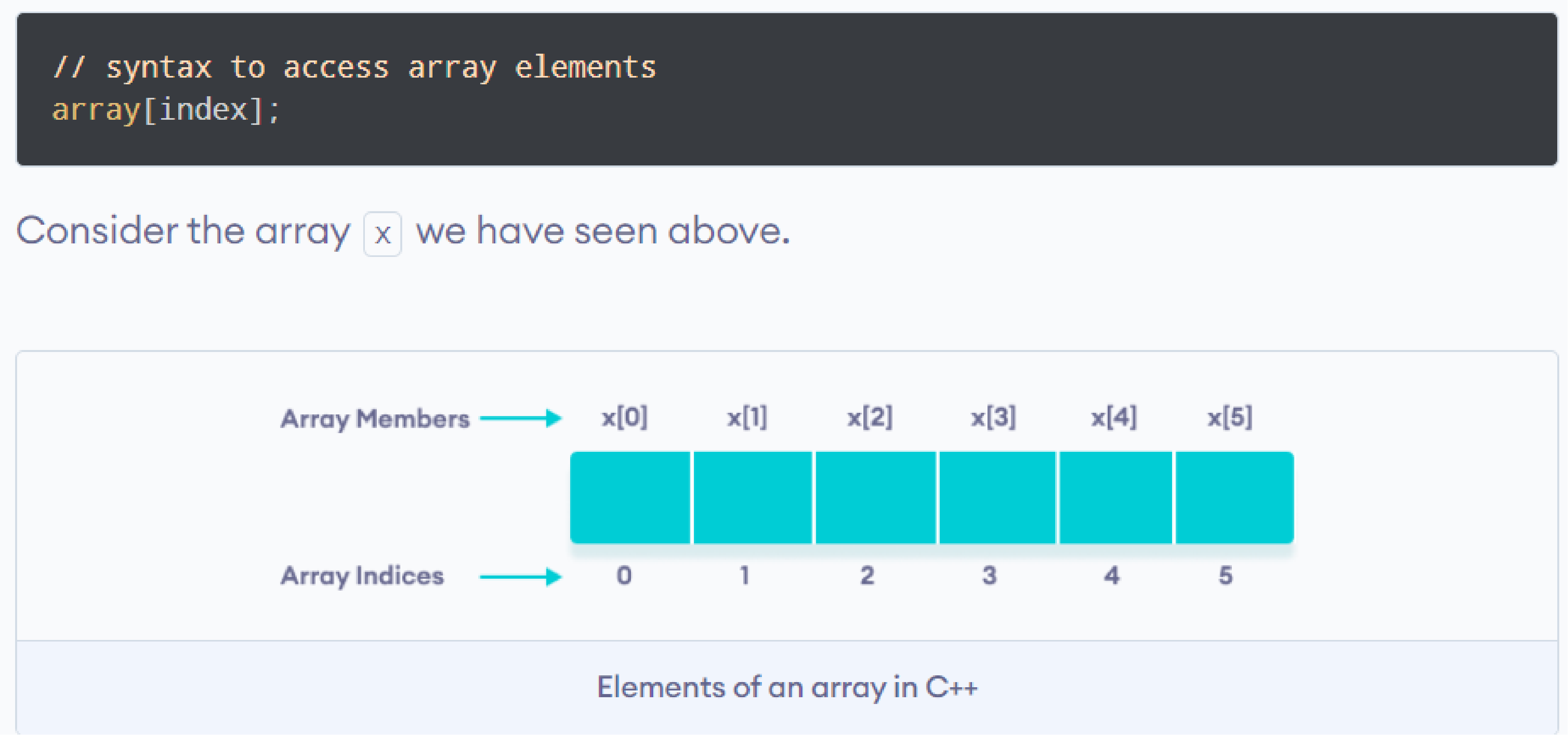
Suppose a class has 30 students, and we need to store the grades of all of them. Instead of creating 30 separate variables, we can simply create an array:

```
double grade[30];
```

Here, grade is an array that can hold a maximum of 30 elements of double type.

In C++, the size and type of arrays cannot be changed after its declaration.

Arrays

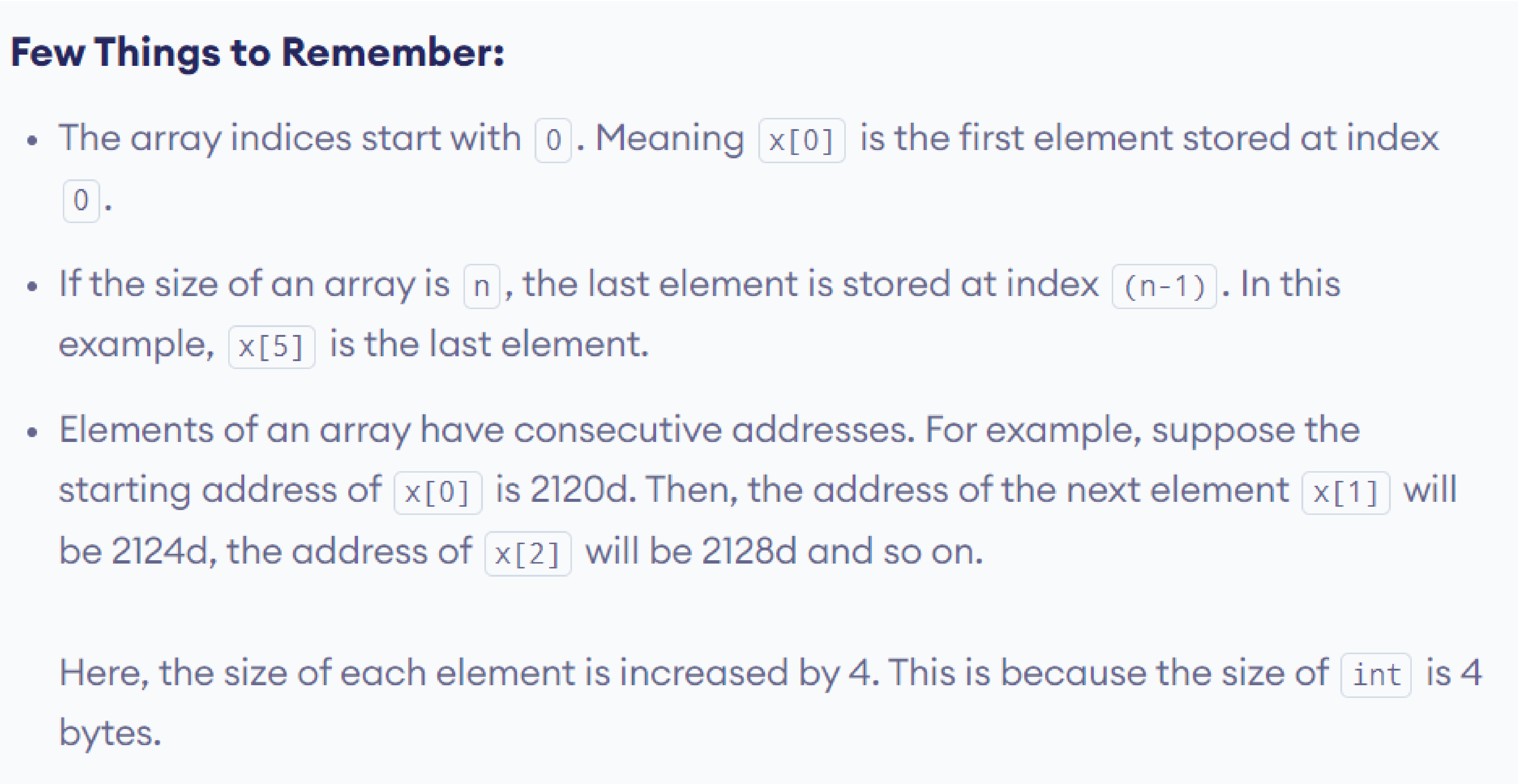


```
// syntax to access array elements
array[index];
```

Consider the array `x` we have seen above.

Elements of an array in C++

Arrays



Few Things to Remember:

- The array indices start with `0`. Meaning `x[0]` is the first element stored at index `0`.
- If the size of an array is `n`, the last element is stored at index `(n-1)`. In this example, `x[5]` is the last element.
- Elements of an array have consecutive addresses. For example, suppose the starting address of `x[0]` is 2120d. Then, the address of the next element `x[1]` will be 2124d, the address of `x[2]` will be 2128d and so on.

  Here, the size of each element is increased by 4. This is because the size of `int` is 4 bytes.

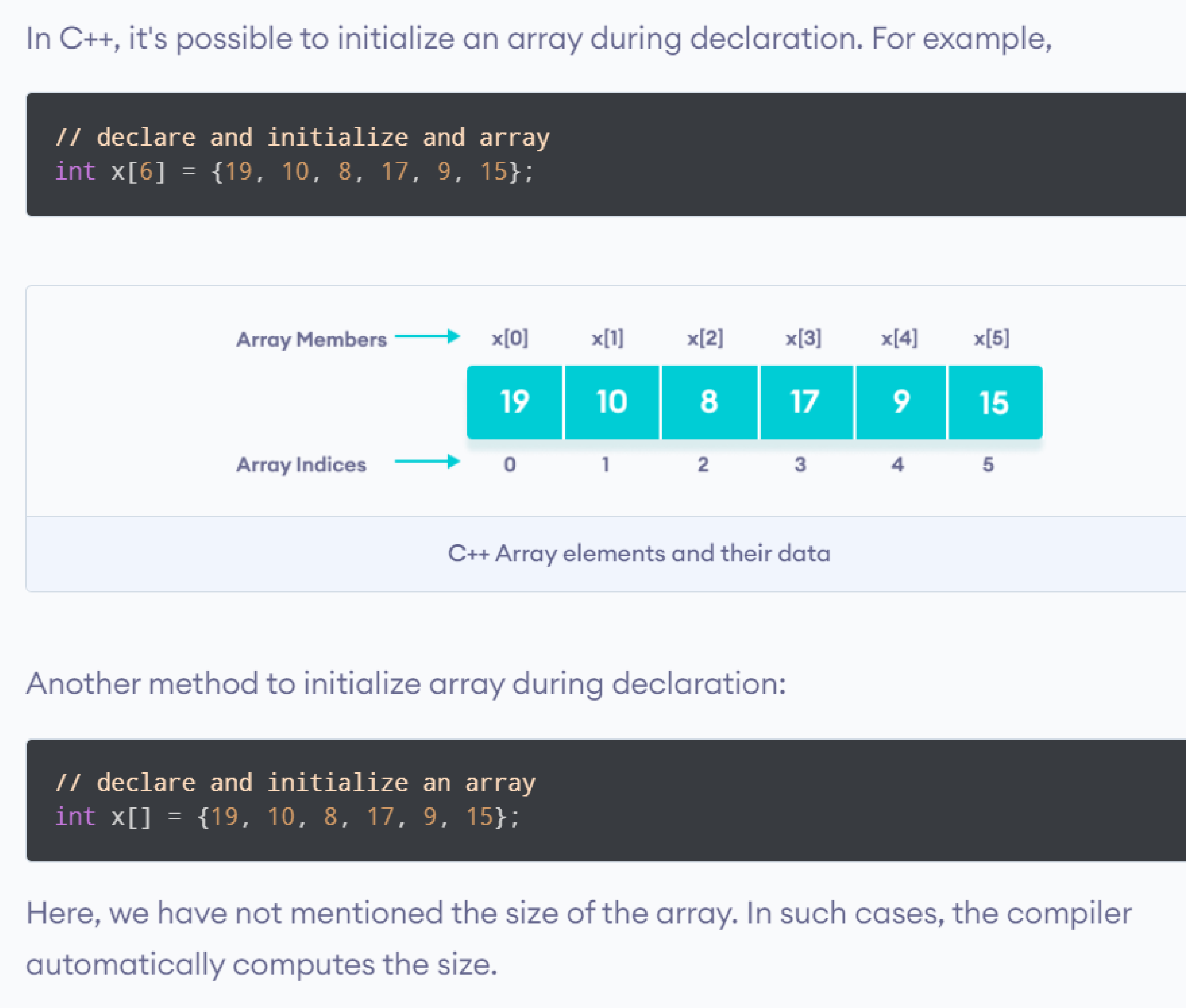
Arrays

Few Things to Remember:

- The array indices start with 0. Meaning x[0] is the first element stored at index 0.
- If the size of an array is n, the last element is stored at index (n-1). In this example, x[5] is the last element.
- Elements of an array have consecutive addresses. For example, suppose the starting address of x[0] is 2120d. Then, the address of the next element x[1] will be 2124d, the address of x[2] will be 2128d and so on.

Here, the size of each element is increased by 4. This is because the size of int is 4 bytes.

Arrays



In C++, it's possible to initialize an array during declaration. For example,

```
// declare and initialize and array
int x[6] = {19, 10, 8, 17, 9, 15};
```

Array Members →	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
	19	10	8	17	9	15
Array Indices →	0	1	2	3	4	5

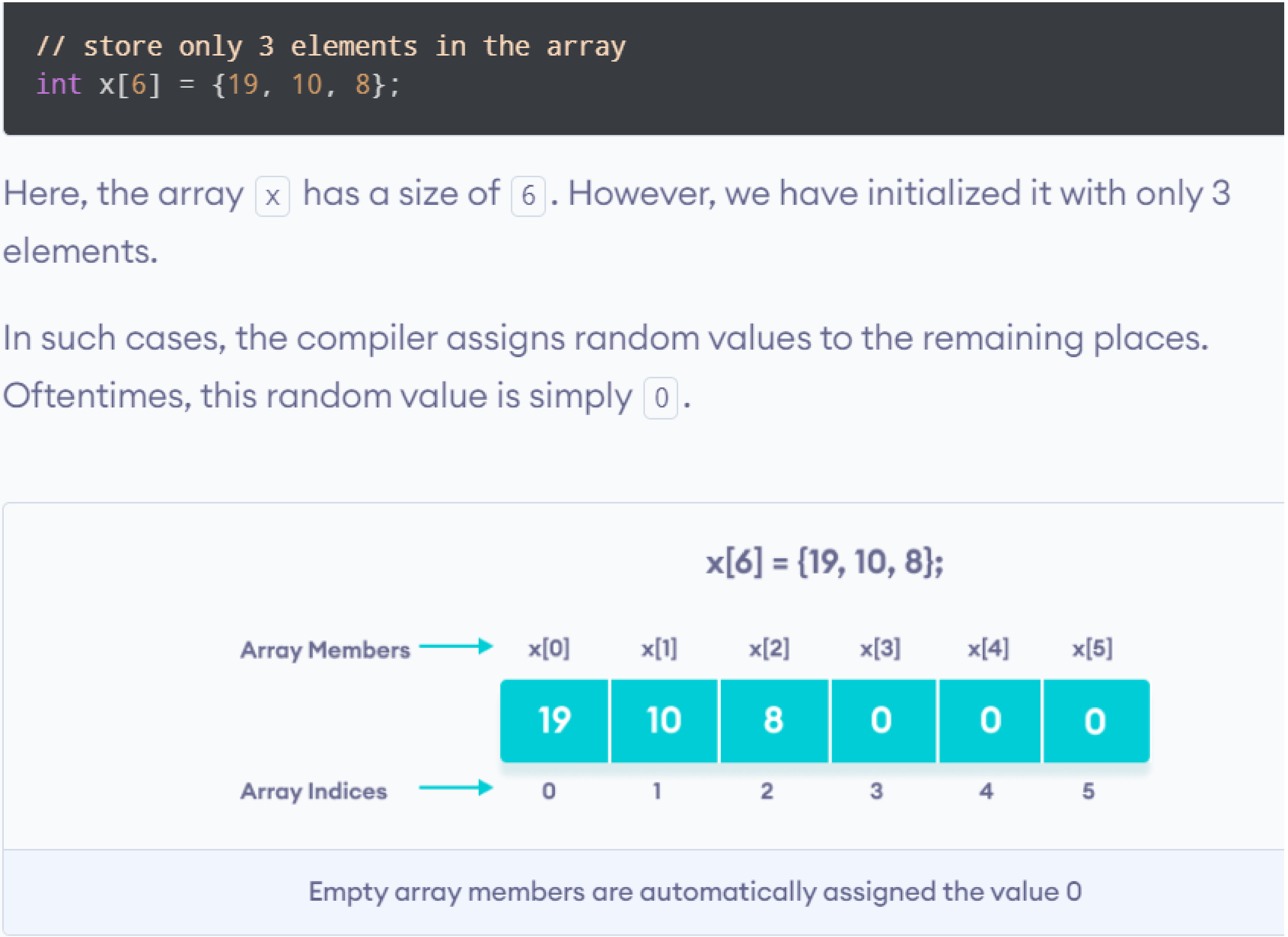
C++ Array elements and their data

Another method to initialize array during declaration:

```
// declare and initialize an array
int x[] = {19, 10, 8, 17, 9, 15};
```

Here, we have not mentioned the size of the array. In such cases, the compiler automatically computes the size.

Arrays



```
// store only 3 elements in the array
int x[6] = {19, 10, 8};
```

Here, the array `x` has a size of `6`. However, we have initialized it with only 3 elements.

In such cases, the compiler assigns random values to the remaining places. Oftentimes, this random value is simply `0`.

x[6] = {19, 10, 8};

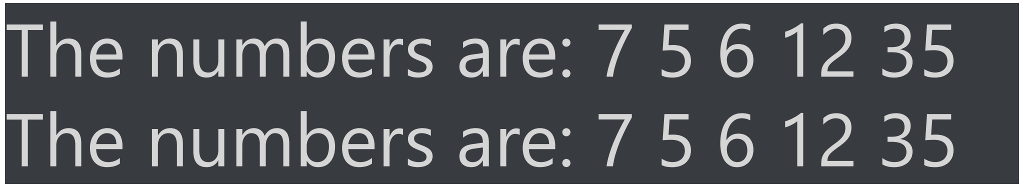
Array Members →	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
	19	10	8	0	0	0
Array Indices →	0	1	2	3	4	5

Empty array members are automatically assigned the value 0

Arrays

Example: Displaying Array Elements

```
#include <iostream>
using namespace std;
int main() {
    int numbers[5] = {7, 5, 6, 12, 35};
    cout << "The numbers are: ";
    //  Printing array elements,  using range based for loop
    for (const int &n : numbers) {
        cout << n << "  ";
    }
    cout << “ \nThe numbers are: ";
    //  Printing array elements,  using traditional for loop
    for (int i = 0; i < 5; ++i) {
        cout << numbers[i] << "  ";
    }
    return 0;
}
```

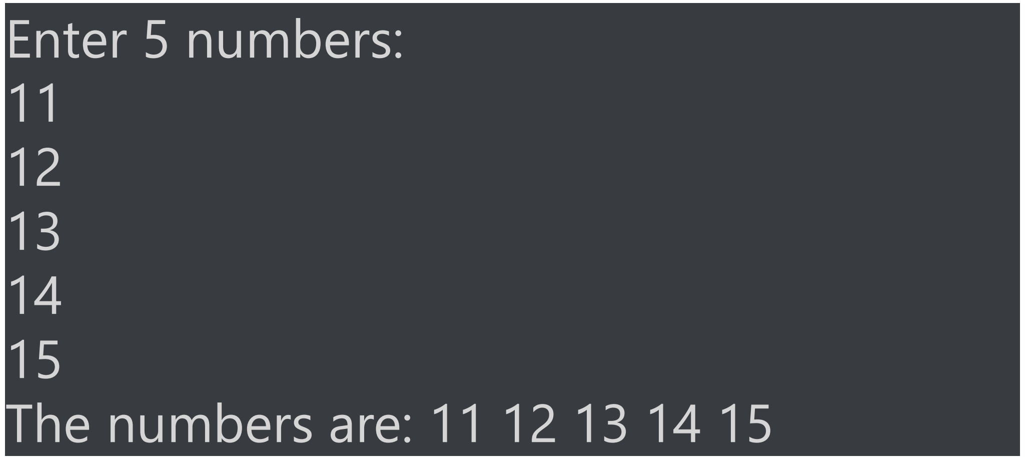


```
The numbers are: 7 5 6 12 35
The numbers are: 7 5 6 12 35
```

Arrays

Example: Take Inputs from User and Store Them in an Array

```
#include <iostream>
using namespace std;
int main() {
    int numbers[5];
    cout << "Enter 5 numbers: " << endl;
    //  store input from user to array
    for (int i = 0; i < 5; ++i) {
        cin >> numbers[i];
    }
    cout << "The numbers are: ";
    //  print array elements
    for (int n = 0; n < 5; ++n) {
        cout << numbers[n] << "  ";
    }
    return 0;
}
```

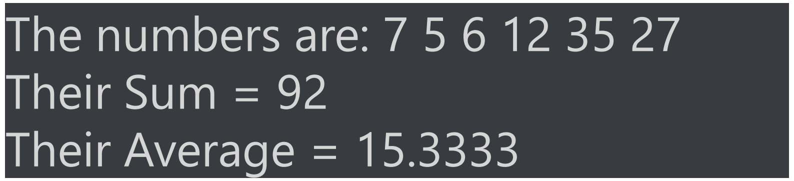


```
Enter 5 numbers:
11
12
13
14
15
The numbers are: 11 12 13 14 15
```

Arrays

Example: Display Sum and Average of Array Elements Using for Loop

```
#include <iostream>
using namespace std;
int main() {
double numbers[ ] = {7, 5, 6, 12, 35, 27}; // initialize an array without specifying size
    double sum = 0;
    double count = 0;
    double average;
    cout << "The numbers are: ";
    //  print array elements ,  &  use of range-based for loop
    for (const double &n : numbers) {
        cout << n << "  ";
        sum += n; //  calculate the sum,  & count the no. of array elements
        ++count;
    }
cout << "\nTheir Sum = " << sum << endl; // print the sum
    average = sum / count; // find the average
    cout << "Their Average = " << average << endl;
    return 0;
}
```



```
The numbers are: 7 5 6 12 35 27
Their Sum = 92
Their Average = 15.3333
```

End of
Lecture # 10