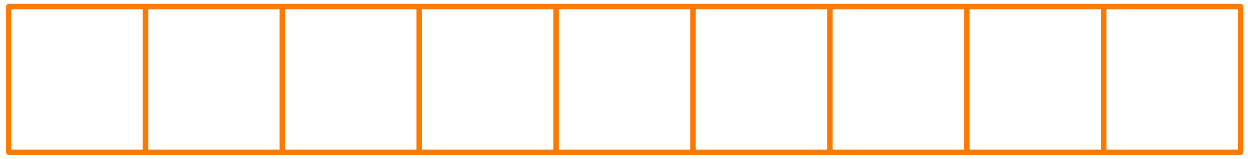
Arrays

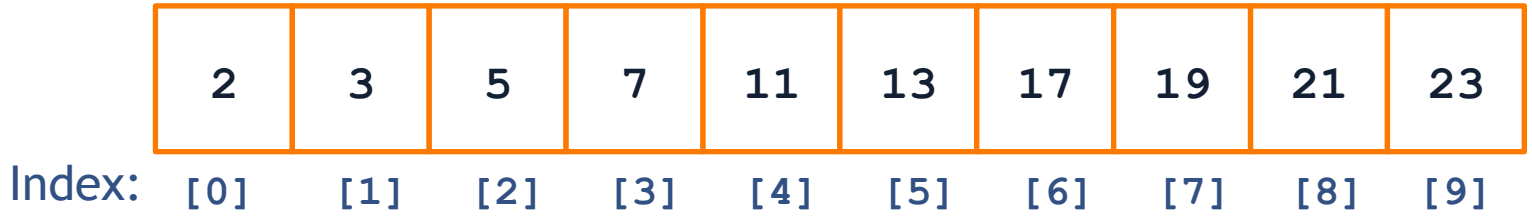
Prof. Wade Fagen-Ulmschneider

An array stores data in **blocks of sequential memory:**

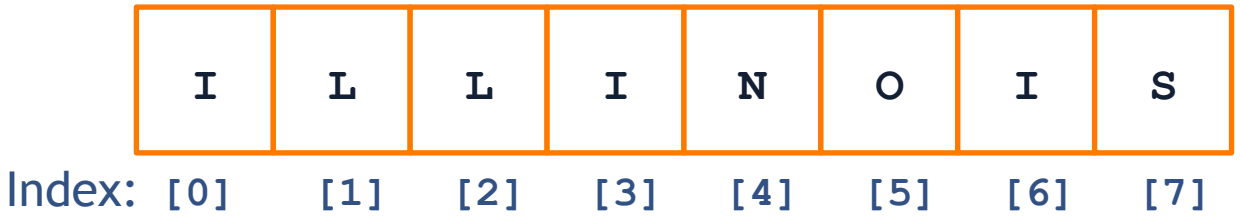


Examples:

- *An array of the first ten prime numbers:*



- *An array of eight characters:*

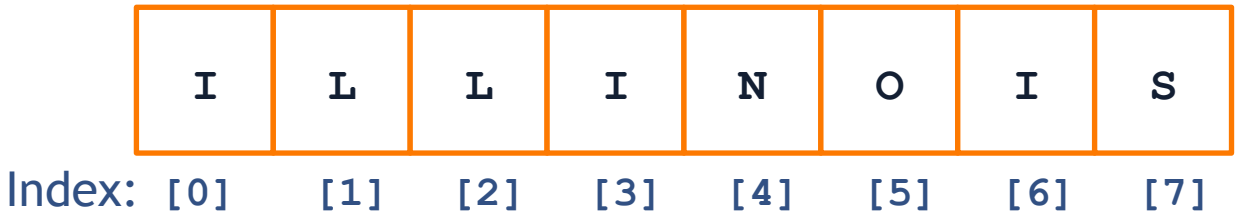


arrays/ex1/main.cpp

```cpp
#include <iostream>

int main() {
  // Create a fixed-sized array of 10 primes:
  int values[10] = { 2, 3, 5, 7, 11, 13, 15, 17, 21, 23 };

  // Outputs the 4th (index 3) prime:
  std::cout << values[3] << std::endl;

  return 0;
}
```

Array Limitation #1

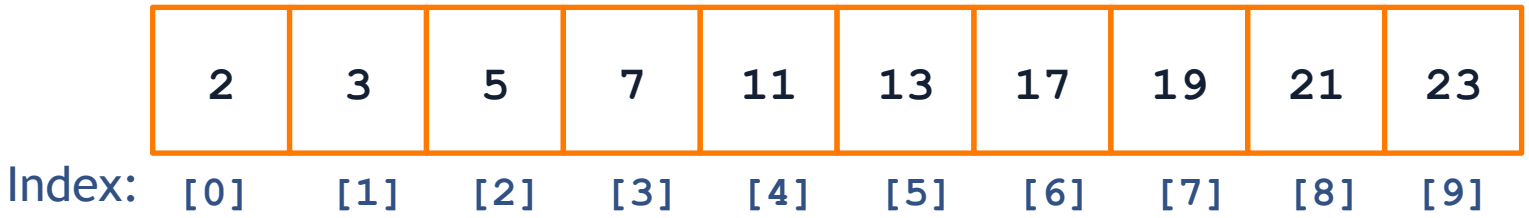
- All data in an array must be of the same type:
  - An integer array must only contain integers.
  - A string array must only contain strings.
  - …



We know two facts about arrays:

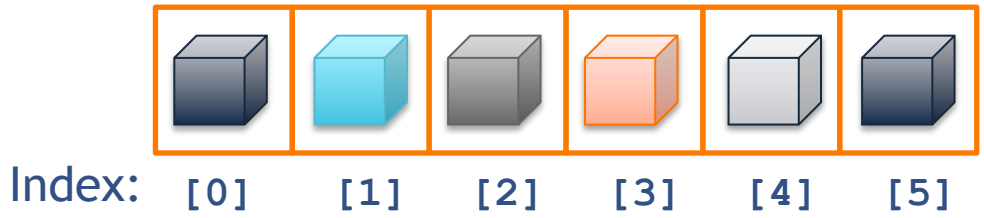
1. Elements are **all the same type**.
2. The **size** (number of bytes) of the type of data is known.

We can calculate the offset to any given index from the start of the array:



Example:

- Consider an array of six `Cube` objects:



  - Using `sizeof(Cube)`, we found each cube is 8 bytes large.

- What is the offset to index 3?

arrays/ex2/main.cpp

```cpp
#include <iostream>

int main() {
  // Create an array of 10 primes:
  int values[10] = { 2, 3, 5, 7, 11, 13, 15, 17, 21, 23 };

  // Print the size of each type `int`:
  std::cout << sizeof(int) << std::endl;

  // Using pointer arithmetic, ask the computer to calculate
  // the offset from the beginning of the array to [2]:
  int offset = (long)&(values[2]) - (long)&(values[0]);
  std::cout << offset << std::endl;

  return 0;
}
```

arrays/ex3/main.cpp

```
13 int main() {
14   // Create an array of 3 `Cube`s:
15   Cube cubes[3] = { Cube(11), Cube(42), Cube(400) };
16
17
18
19   // Print the size of each type `Cube`:
20   std::cout << sizeof(Cube) << std::endl;
21
22   // Using pointer arithmetic, ask the computer to calculate
23   // the offset from the beginning of the array to [2]:
24   int offset = (long)&(cubes[2]) - (long)&(cubes[0]);
25   std::cout << offset << std::endl;
26
27   return 0;
28 }
```

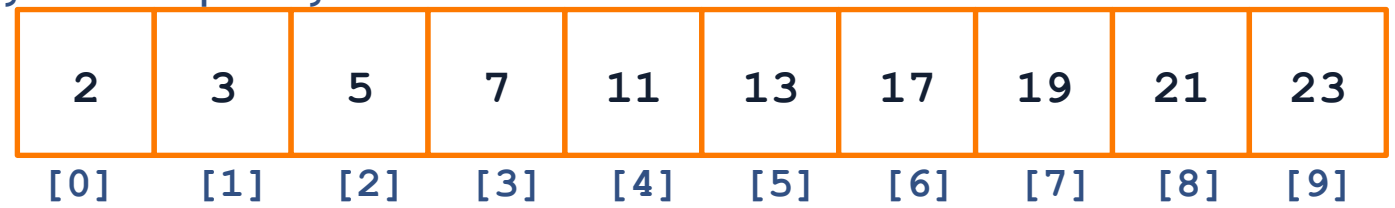
Array Limitation #2

- Arrays have a fixed **capacity**.
  - Arrays **must** store their data sequentially in memory.
  - The **capacity** of an array is the maximum number of elements that can be stored.
  - The **size** of an array is the current number of elements stored in the array.

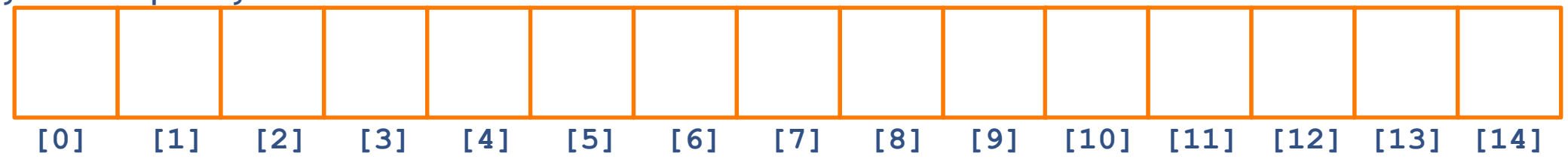
An array is full when the **size** of the array is equal to the **capacity**.

The only way to add another element is to allocate a new, larger array and copy over all of the data:

Array with capacity=10:



Array with capacity=15:



std::vector

The **std::vector** implements an array that dynamically grows in size automatically. However, all properties remain true:

- Array elements are a fixed data type.
- At any given point, the array has a fixed capacity.
- The array must be expanded when the capacity is reached.

arrays/ex4/main.cpp

```
15    std::vector<Cube> cubes{ Cube(11), Cube(42), Cube(400) };
16
17    // Examine capacity:
18    std::cout << "Initial Capacity: " << cubes.capacity() << std::endl;
19    cubes.push_back( Cube(800) );
20    std::cout << "Size after adding: " << cubes.size() << std::endl;
21    std::cout << "Capacity after adding: " << cubes.capacity() << std::endl;
22
23    // Using pointer arithmetic, ask the computer to calculate
24    // the offset from the beginning of the array to [2]:
25    int offset = (long)&(cubes[2]) - (long)&(cubes[0]);
26    std::cout << "Memory separation: " << offset << std::endl;
27
28    // Find a specific `target` cube in the array:
29    Cube target = Cube(400);
30    for (unsigned i = 0; i < cubes.size(); i++) {
31      if (target == cubes[i]) {
32        std::cout << "Found target at [" << i << "]" << std::endl;
33      }
34    }
```