Functions and Arrays in ANSI C

Modularity, Memory Management and Best Practices

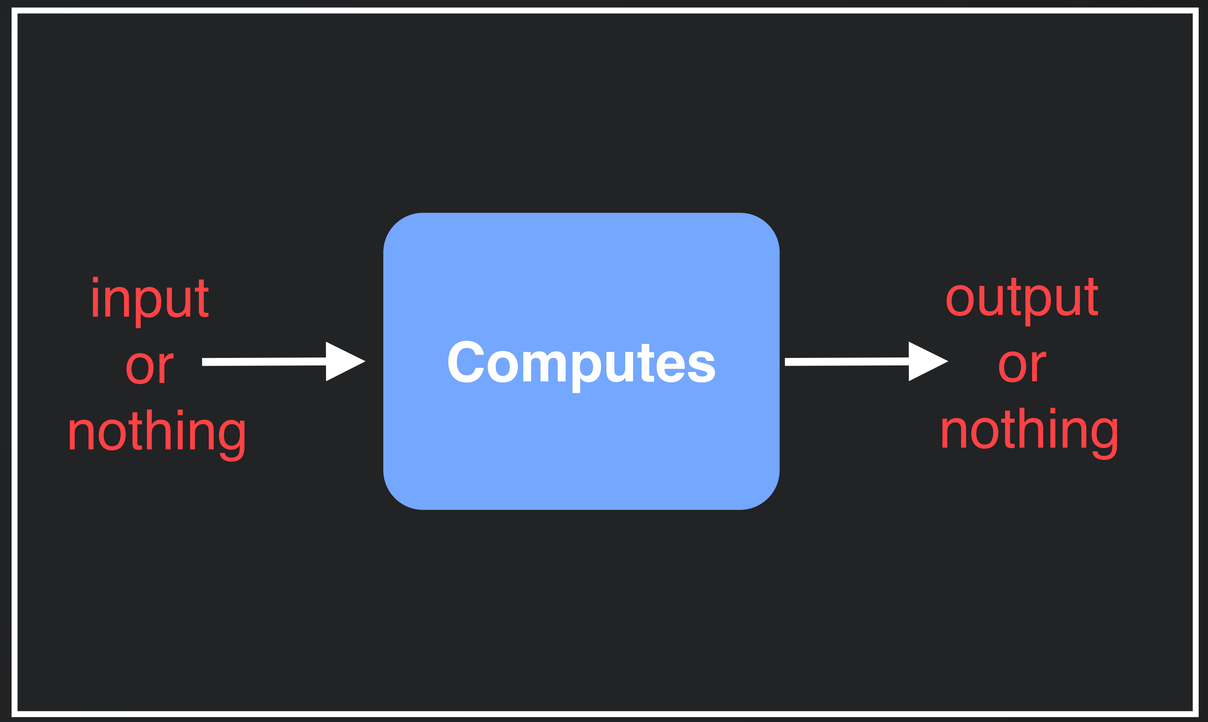
Alex Giulio Berton

What is a Function?

A "black box" that accepts input, processes it, and returns an output.

- ✓ **Abstraction:** Hides complex details.
- ✓ **Input (Parameters):** Data needed to work.
- ✓ **Output (Return):** The result of the processing.

Allows applying the DRY principle: *Don't Repeat Yourself*.



Function Syntax

1. Prototype (Declaration)

Informs the compiler that the function exists. Goes before `main`.

```
float average(int a, int b);
// Note the final semicolon!
```

2. Definition (Implementation)

The actual code that performs the task.

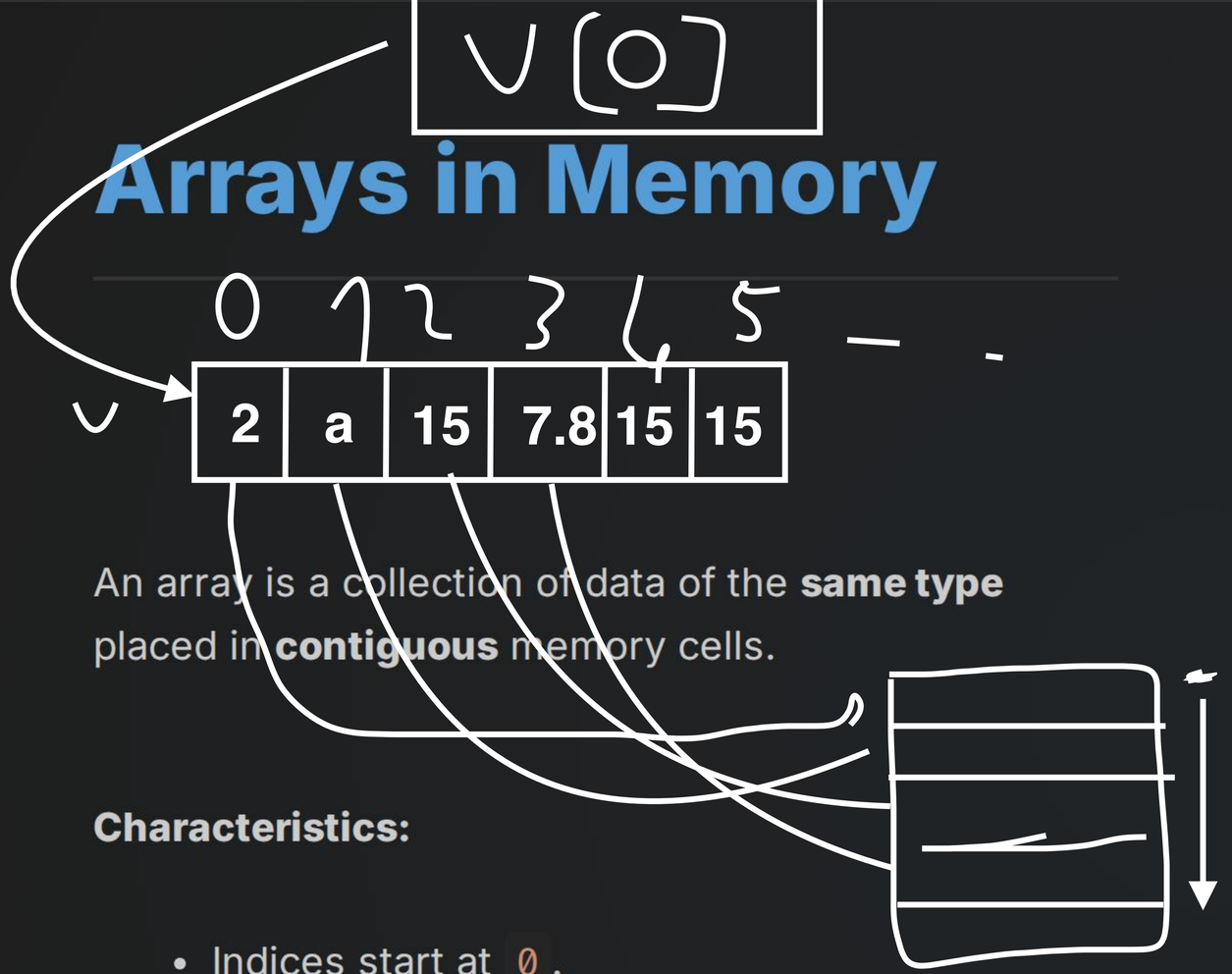
```
float average(int a, int b) {
    float res = (a + b) / 2.0;
    return res;
}
```

Pass by Value

- **The Copy:** In C, simple variables (int, char, float) are passed by value.
- **Isolation:** The function works on a local copy. Modifying the parameter **does not** change the original variable in `main`.
- **Memory:** New memory cells are allocated for formal parameters.

> **Note:** This does NOT apply to arrays, which behave like pointers.

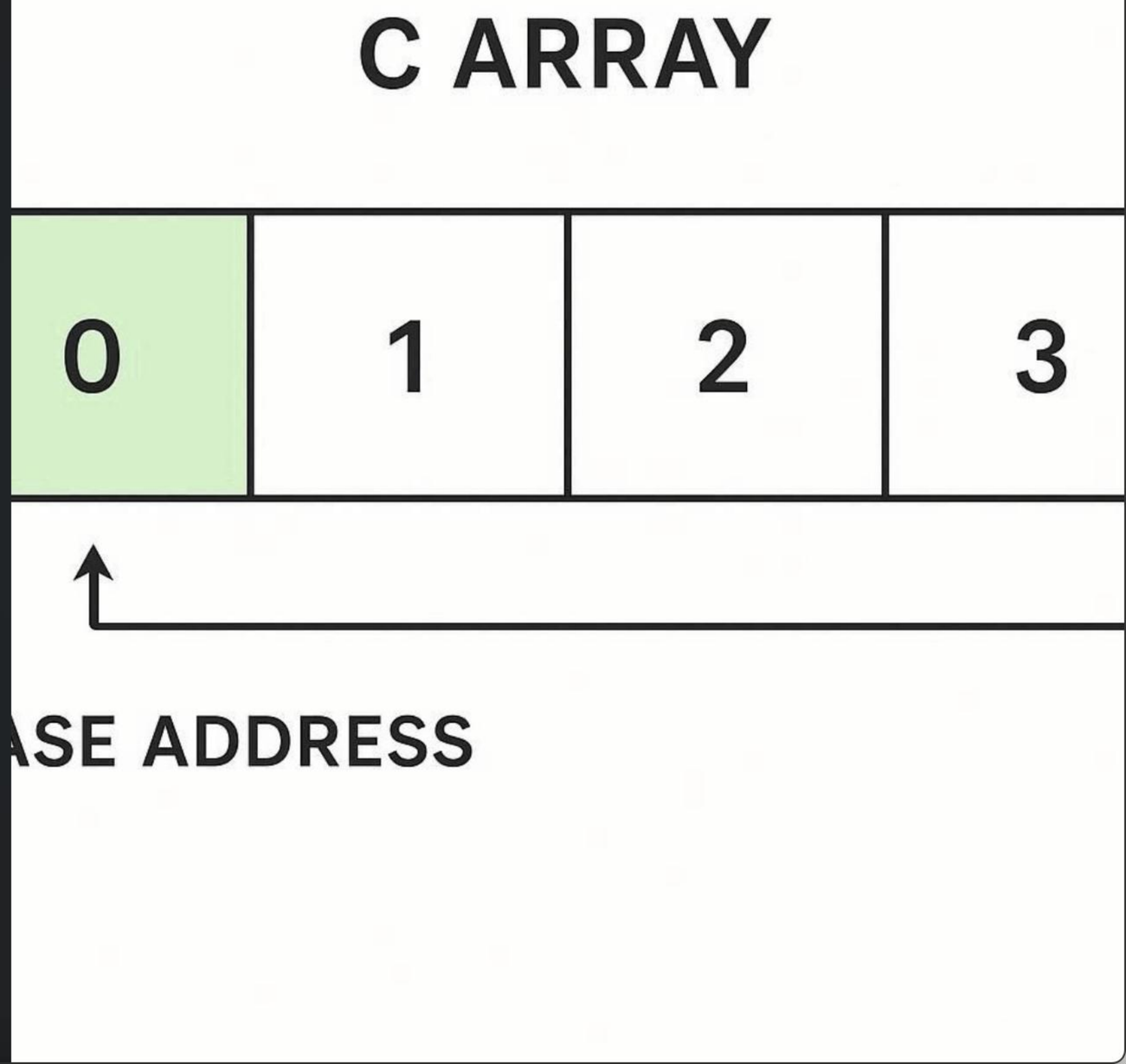
Arrays in Memory



An array is a collection of data of the **same type** placed in **contiguous** memory cells.

Characteristics:

- Indices start at `0`.
- Instant access (Random Access) via index.
- Fixed size defined at compilation.



Declaration and Access

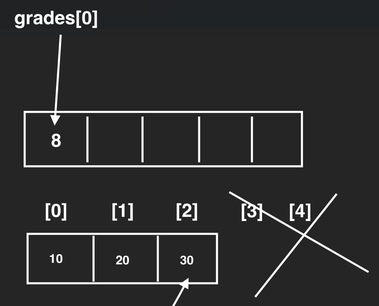
Correct Syntax

```
// Declaration
int grades[5];
// Quick initialization
int num[] = {10, 20, 30};

// Read/Write
grades[0] = 8;
int x = num[2]; // x is 30
```

type of the internal values

name of the array

number of cells the array is composed of



num[3] = 40; ← Out of bound exception

Common Errors

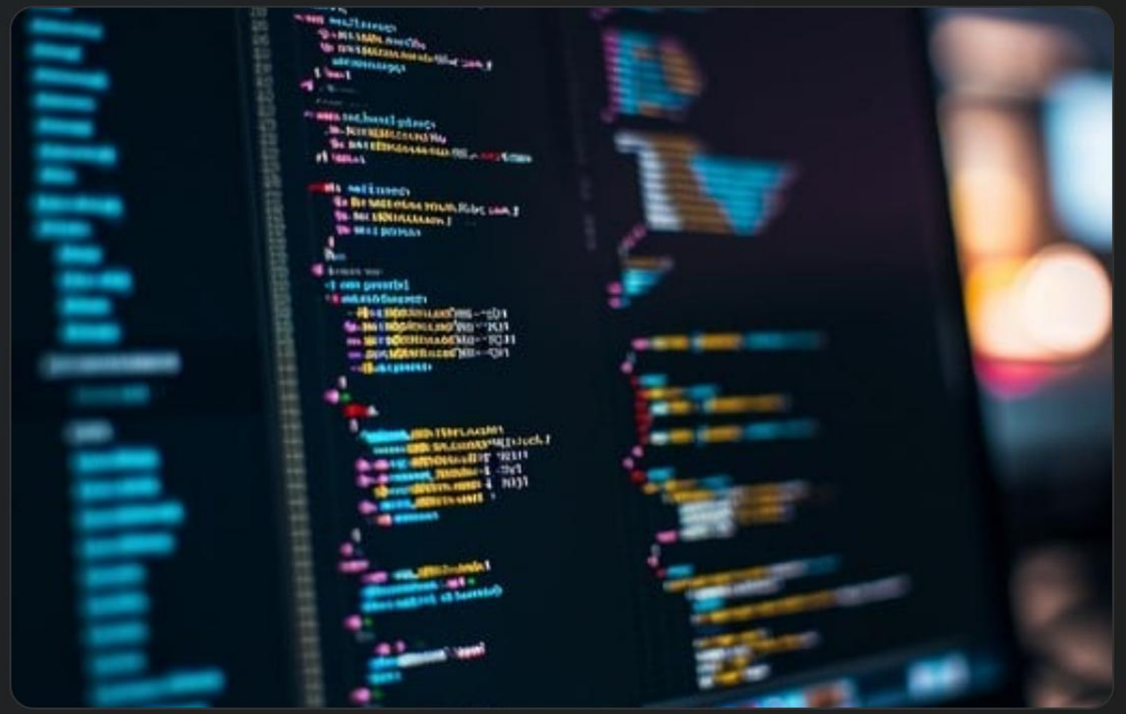
```
int arr[3];

// ERROR: Index out of bounds
arr[3] = 5;
// Valid indices are 0, 1, 2!

// ERROR: Direct assignment
arr = {1, 2, 3};
// Can only be done at initialization
```

Arrays and Functions

When you pass an array to a function, you pass the **address** of the first element (pointer).

Fundamental Rules:

- The array "decays" to a pointer. The whole array is not copied!
- You must **always** pass the size as an extra parameter.
- Modifications made inside the function are **permanent** (Side Effect).



Complete Example: Array Sum

```c
// Function receiving the array
int sum_elements(int arr[], int size) {
    int total = 0;

    // Use 'size' to stop the loop
    for(int i=0; i < size; i++) {
        total += arr[i];
    }

    return total;
}

int main() {
    int numbers[] = {5, 10, 15};

    // Pass the array and its length (3)
    int res = sum_elements(numbers, 3);

    printf("The sum is: %d", res);
    return 0;
}
```

Code Analysis

1. **Signature:** `arr[]` indicates we are receiving a pointer to integers.

2. **Size Parameter:** Without `size`, the function wouldn't know when to stop, risking reading "garbage" memory.

3. **Return:** The calculated value (30) is returned to main.