Arrays

C Reference

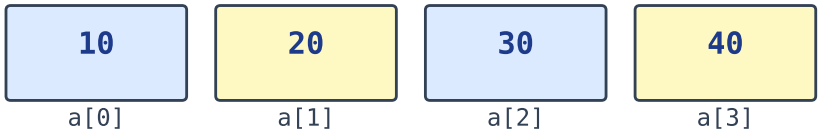
1-D arrays

An array holds several values of one type. Index from 0. C does **not** store an array's length, so a common trick computes the element count: `sizeof(a) / sizeof(a[0])`.

```c
#include <stdio.h>

int main(void) {
    int scores[3] = {88, 71, 95};
    printf("%d\n", scores[0]);     // 88
    scores[1] = 100;
    printf("%d\n", scores[1]);     // 100
    int n = sizeof(scores) / sizeof(scores[0]);
    printf("%d\n", n);             // 3
    return 0;
}
```

`int a[4] = {10, 20, 30, 40};`



Exam tip: indices start at 0; a[i] is one element; length is fixed at creation

Array slots are indexed from 0

Array algorithms

Traverse an array with a `for` loop to find a max, a total, or a count. Always loop from `0` up to `n - 1`.

```c
#include <stdio.h>

int main(void) {
    int a[] = {3, 9, 2, 7};
    int n = sizeof(a) / sizeof(a[0]);
    int max = a[0], total = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] > max) max = a[i];
        total += a[i];
    }
    printf("%d %d\n", max, total);   // 9 21
    return 0;
}
```

2-D arrays

A 2-D array is a grid of rows and columns : `grid[row][col]`. Use two nested loops to visit every cell.

```c
#include <stdio.h>

int main(void) {
    int grid[2][3] = {{1, 2, 3}, {4, 5, 6}};
    printf("%d\n", grid[1][2]);    // 6
    for (int r = 0; r < 2; r++) {
        for (int c = 0; c < 3; c++) {
            printf("%d ", grid[r][c]);
        }
    }
    printf("\n");                  // 1 2 3 4 5 6
    return 0;
}
```

Common mistakes

- C does NOT check array bounds: writing `a[n]` in a size-`n` array is undefined behaviour.
- Valid indexes run from `0` to `n - 1`.

- In most expressions an array converts to a pointer to its first element, but the array itself is not a pointer. `sizeof(a)` gives the whole byte size for an actual array; an array parameter is adjusted to a pointer, so that function needs a separately supplied length.