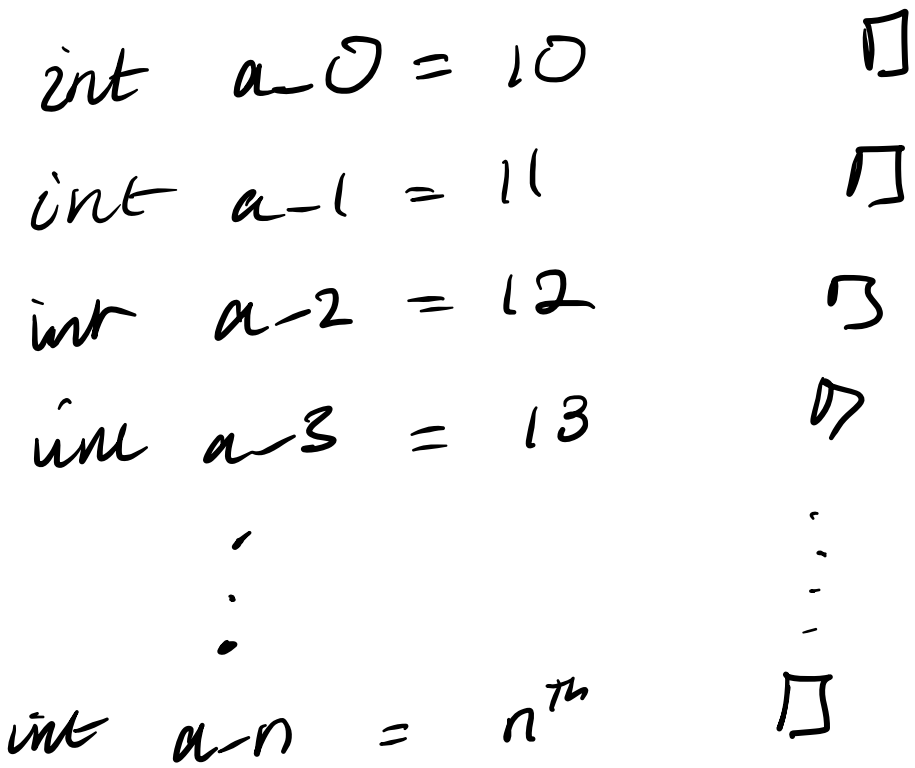
Chapter Seven: Introduction to Arrays

William T. Doan

13 November 2024

Introduction



The Definition of an Array

Definition 7.1.1

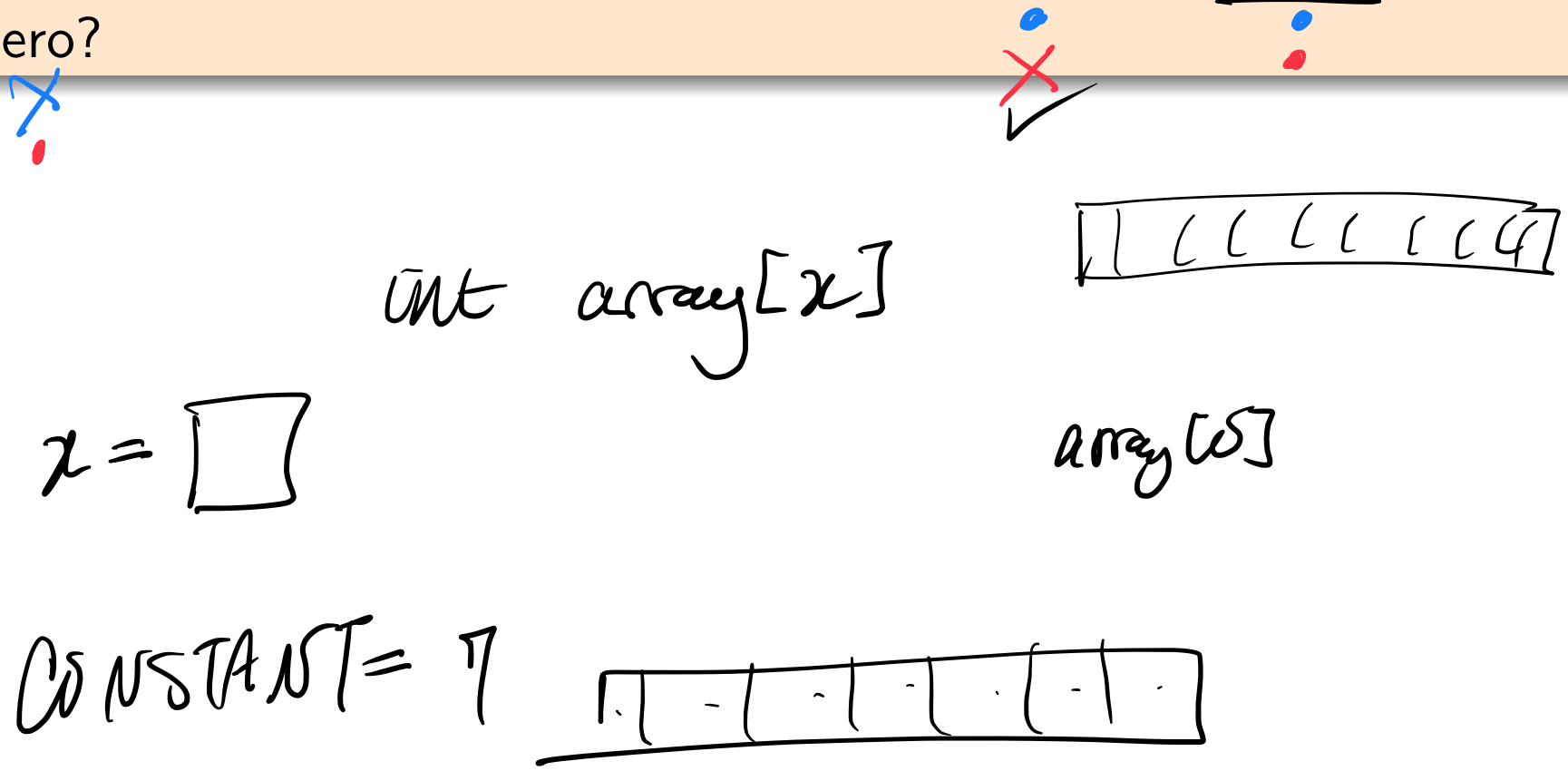
The array is a data structure which can hold multiple elements of the same type consecutively and contiguously.

Terminology

```
int array_test[size]
```

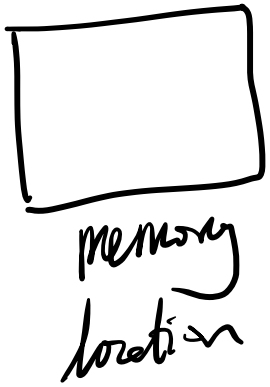
Query 7.1.1

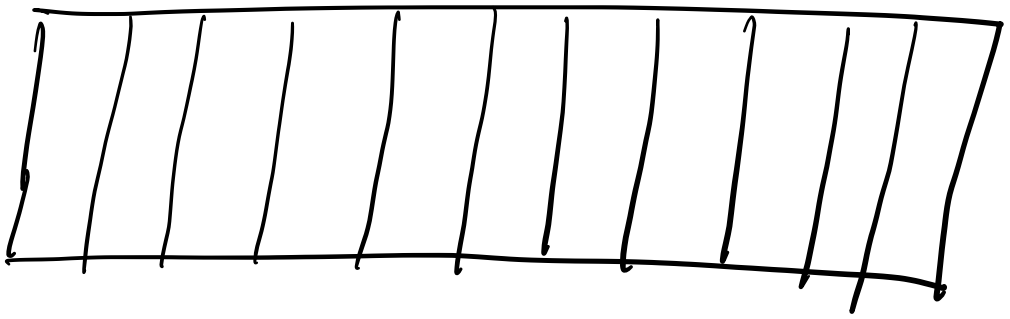
Can the size declarator of the array be a variable? Constant? Can it be zero?



Arrays and Memory

Recall Definition 7.1. Recall Chapter 1.

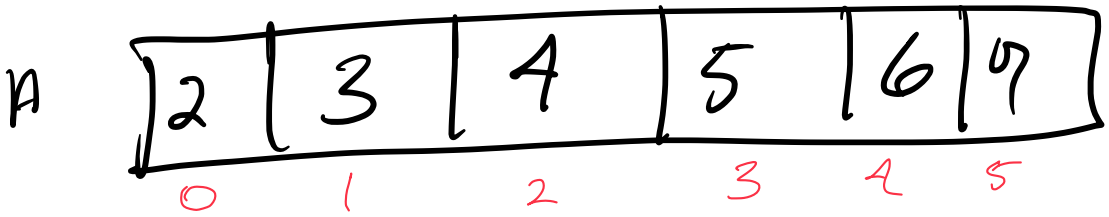




Accessing Arrays in Memory

Query 7.1.2

Are the individual elements of the array accessible? If not, why is that? If so, how?



A[I]

The Definition of an Index (a.k.a., Subscript)

Definition 7.1.2

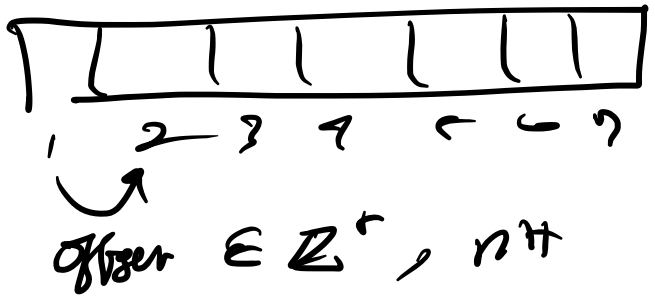
The indices of an array represent the position of each element relative to the start of the array. Each index corresponds to an offset from the starting memory address, allowing access to the specific memory location of each element.

Corollary 7.1.1 to Definition 7.1.2

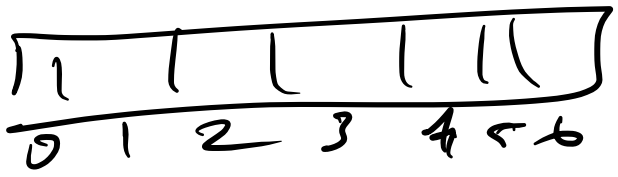
Recall **Definition 7.1.1**. If the array stores variables contiguously in memory, then the index of the array betokens how far from the starting memory address the element is located.

Query 7.1.3

Are the indices of the array zero-based indexed or can they be one-based indexed?



C#, C, Python ...
Zero based indexing.



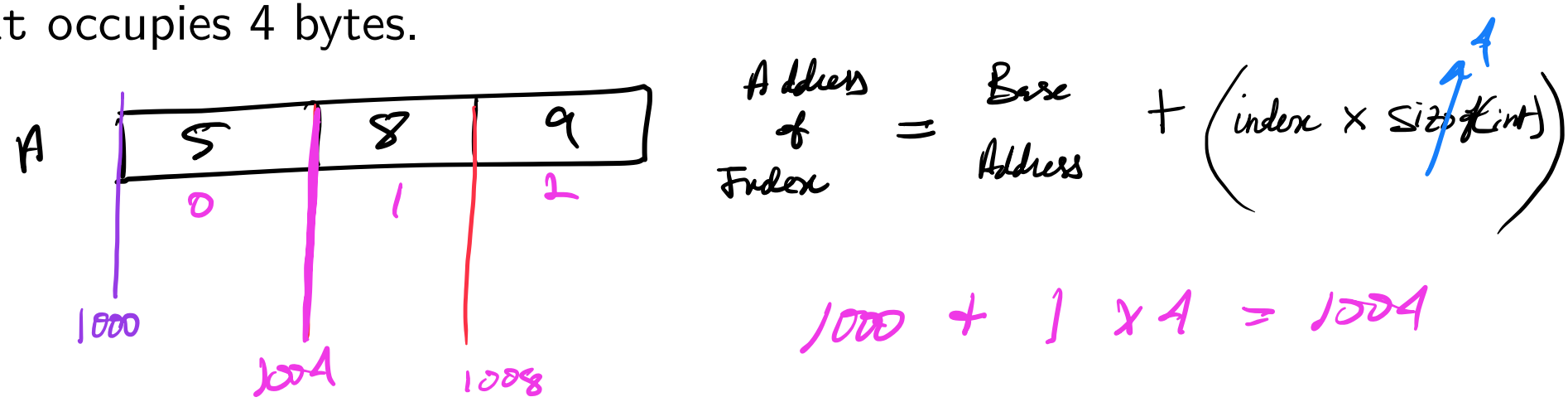
Fortran, Ada,
One-based indexing

Exploring Pointer Arithmetic in Arrays

Suppose we have an integer array defined as:

```
int numbers[3];
```

Let us assume the array `numbers` begins at address 1000 and that each `int` occupies 4 bytes.



Each address calculation illustrates how the base address (1000) offsets by 4 bytes per element. Pointer arithmetic provides efficient, direct access to each element in memory.

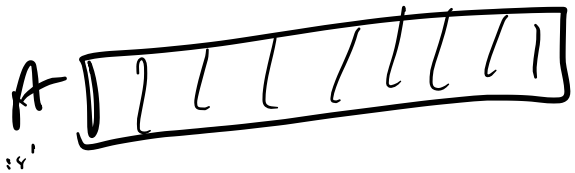
Accessing the Elements of an Array

Query 7.1.4

Can operations be done as an aggregate on an array? What conclusions can be drawn from that?

std::cout << A << std::endl;

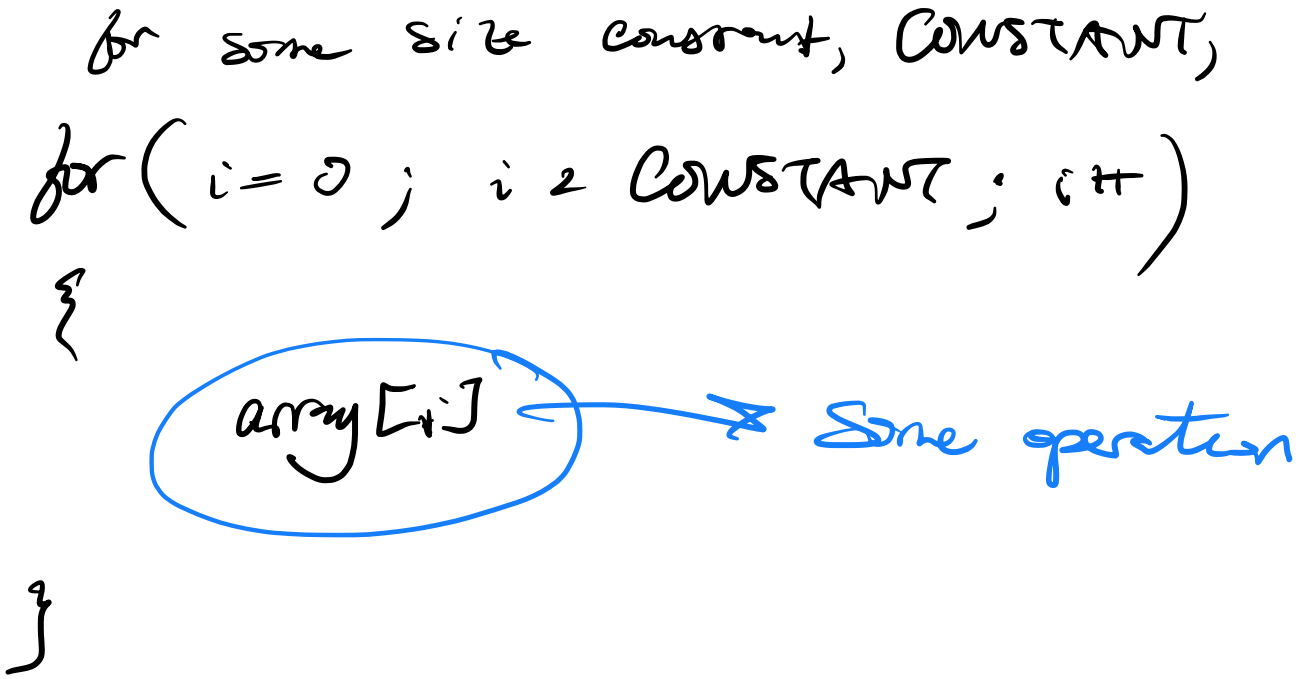
↳ error



Each element of an array can be accessed individually.

Query 7.1.5

How might all the elements on an array be accessed?

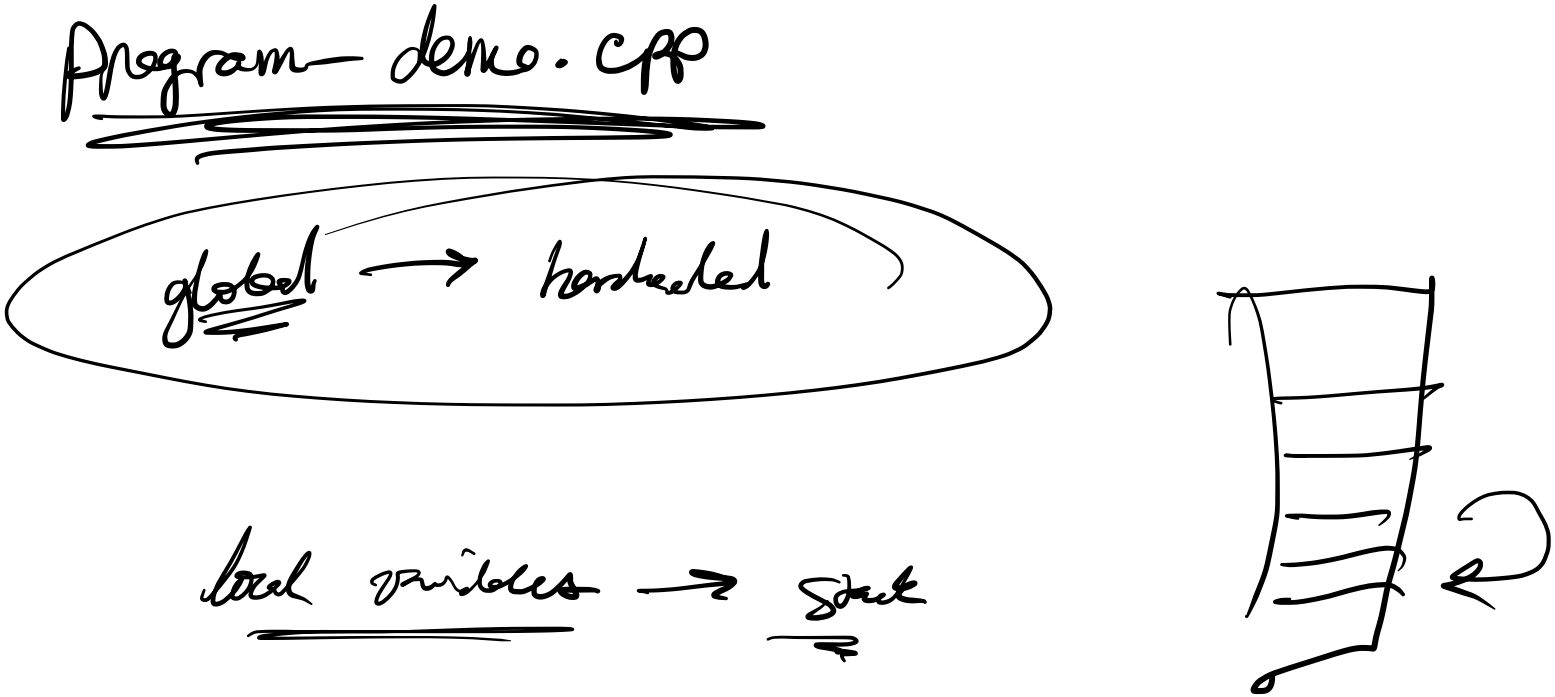


Default Array Initialization

Elements of **local arrays** will be left uninitialized whereas those of **global arrays** are initialized to 0.

Query 7.1.6

Why is that?



On Array Initialization

Let there be A.

A single index of an array can be initialized.

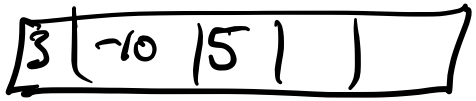
A[5] ← ——

Elements of an array can be initialized using a list.

A[5] = {3, -10, 5, 7, 13}

A[ ] = {3, -10, 5, 7, 13}

An array does not have to be fully initialized.

A[5] = {3, -10, 5, , } → A 

Query 7.1.6

Let `int numbers[7] = {9, 2, , 7, -2, , 19};`. Is that valid?

It is possible to define an array without specifying its size, provided there exists an initialization list.

You must provide either an array size declarator or an initialization list when defining an array.

On Bounds Checking in C++

The C++ compiler does not perform array bounds checking. This means a statement that uses an array subscript outside the bounds of the array will not be caught by the compiler.

for(i ; i <= CONST; i++)

A[5] ⟶ A[6]

From this, there arises certain consequences thus,

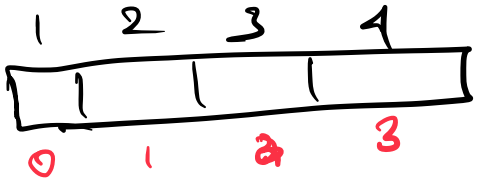
- Corrupt other variables;
- Corrupt the stack;
- OS might terminate the program.

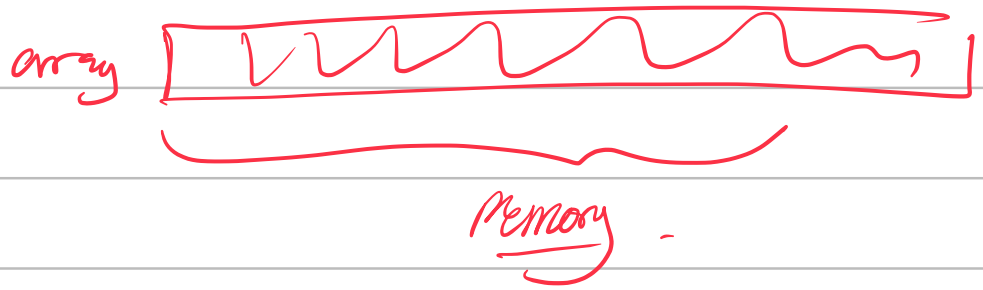
In working with arrays, a common type of mistake is an **off-by-one error**.

for (i ; i < CONST ; i++)

Zero - Based Indexed.

A ← n elements ⟹ n-1 indices.





std::cin >> AAA ... AAA ... AAA ... AAA ...

→ overflow to the buffer

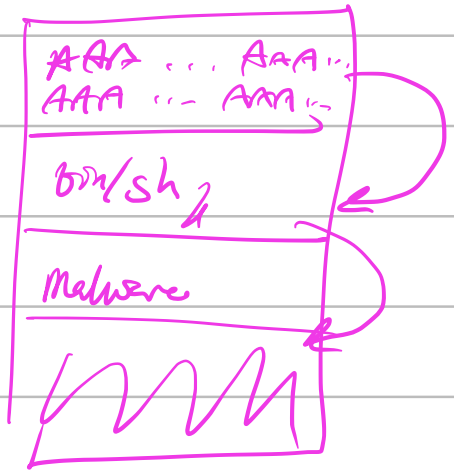
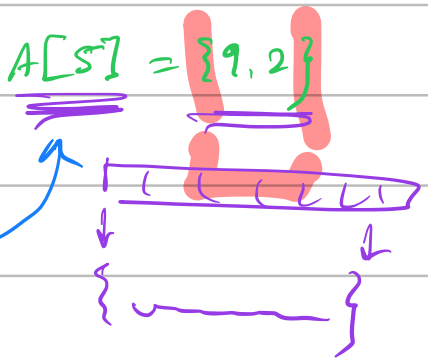
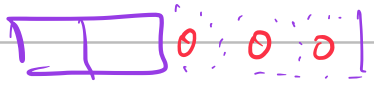


Fig 1.





This is possible — the rest of the elements are initialized to zero.

- garbage comes from previous data — recall the mailbox analogy.