

CSCI 2410 Systems Programming

Fall 2026

Instructor: Jie Zhou

Department of Computer Science
George Washington University



Slides materials are mostly credited to Sabin Mohan and Gabriel Parmer at GWU.

Type Modifiers

- `unsigned`: variables that cannot be negative
- `signed`: signed variable. `char`, `int`, `long` all default to signed.
- `static`: this variable/function are only accessible inside its source file.
- `const`: an immutable value

Type Modifiers

- `unsigned`: variables that cannot be negative
- `signed`: signed variable. `char`, `int`, `long` all default to signed.
- `static`: this variable/function are only accessible inside its source file.
- **`const`: an immutable value**

Pointer Modifier

- `double d` // a regular double
- `double *p_d` // a pointer to a double
- `const double d` // a constant a double
- `const double *d` // a pointer to a double that is constant
- `double* const c_pd` // a constant pointer to a double
- `const double* const c_pd` // a constant pointer to a double that is constant

Arrays

- We have seen arrays before...string!

day_of_week →

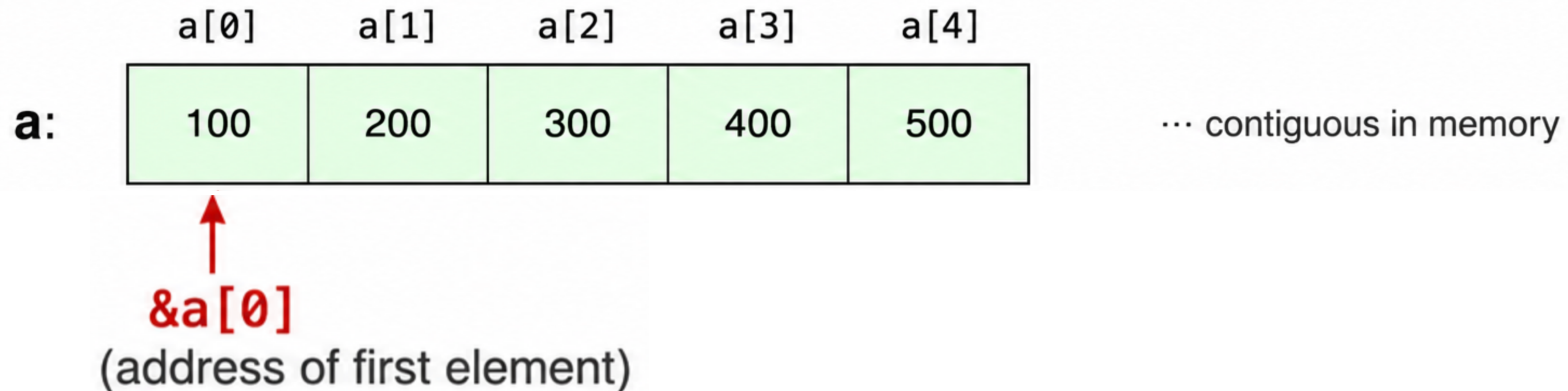
t	h	u	r	s	d	a	y	\0	\0	\0
----------	----------	----------	----------	----------	----------	----------	----------	-----------	-----------	-----------

- Array: a set of *contiguous* data items of the same *type*.

Array Example

```
1  int a[5] = { 100, 200, 300, 400, 500 } ;
```

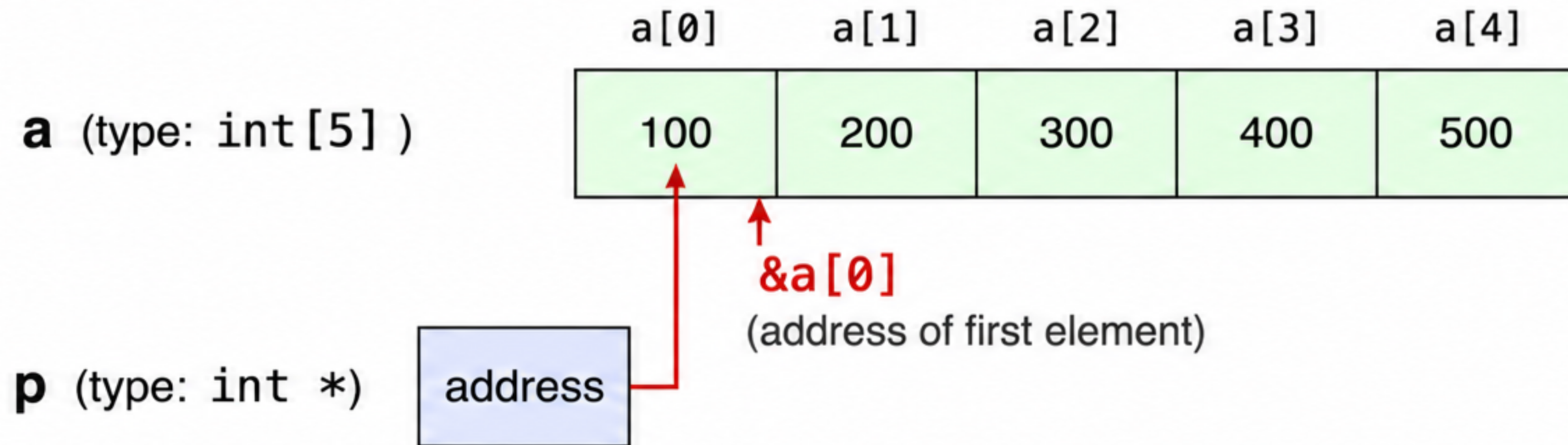
This is what it looks like in memory:



Array vs. Pointer

- An array is *not* a pointer.
- But an array can decay into a pointer to its first element.

```
int a[5] = {100, 200, 300, 400, 500};  
int *p = a;
```



Array vs. Pointer

- The variable name of an array is *not* a pointers.
- But an array's variable can decay into a pointer.

```
int a[5] = {100, 200, 300, 400, 500};  
int *p = a;
```

- When an array item is accessed, e.g., `a[2]`, C is basically doing pointer arithmetic. So, `a[2] → *(a+2)`

<code>a</code>	Start address of the array
<code>a + 2</code>	Address of the third item in the address
<code>*(a + 2)</code>	dereference address (<code>a+2</code>), i.e., accessing the third item in the array

Array vs. Pointer

- The variable name of an array is *not* a pointers.
- But an array's variable can decay into a pointer.

```
int a[5] = {100, 200, 300, 400, 500};  
int *p = a;
```

Array a

Is the actual block of elements

Type: `int [5]`; array size is part of its type.

Storage contains 5 elements

Cannot be assigned: `a = ...`

`sizeof(a) == 5 * sizeof(int)`

Pointer p

Is the address of the array

Type: `int *`. does not know its size

Storage contains one pointer value

Can be assigned: `p = ...`

`sizeof(p) = pointer size`
