Week 5

XMUT-NWEN 241 - 2024 T2

Systems Programming

Mohammad Nekooei

School of Engineering and Computer Science

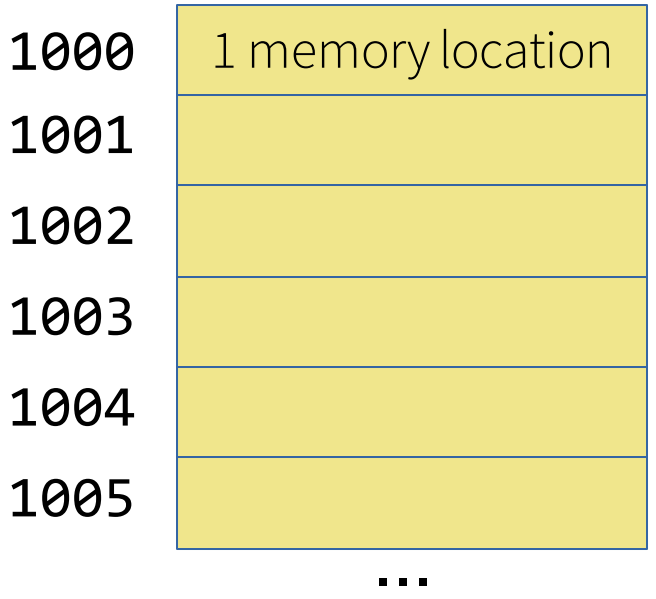
Victoria University of Wellington

Pointers

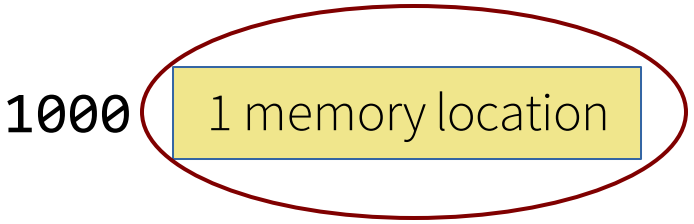
Memory Location

- All information accessible to a running computer program are stored somewhere in the computer's memory

Every *memory location* is identified by an **address**



Memory Location



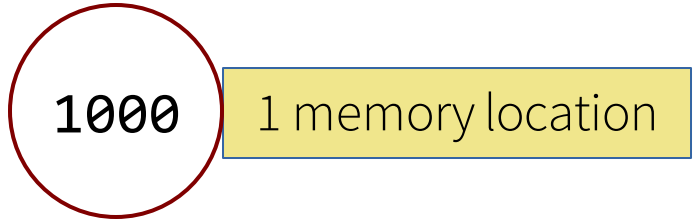
- How big is 1 memory location?
  - It depends on the computer memory architecture

Word-addressable architecture:
- Every memory location corresponds to one *word*

Byte-addressable architecture:
- Every memory location corresponds to one *byte*

Most computers today have byte-addressable memory

Memory Location



- How big is the address?
  - It depends on the number of bits used by CPU for addressing

- Example:
  - In a computer that uses 32 bits for addressing, an address has 32 bits
  - If the computer has byte-addressable memory, then the memory space is 2^{32} bytes = 4 gigabytes

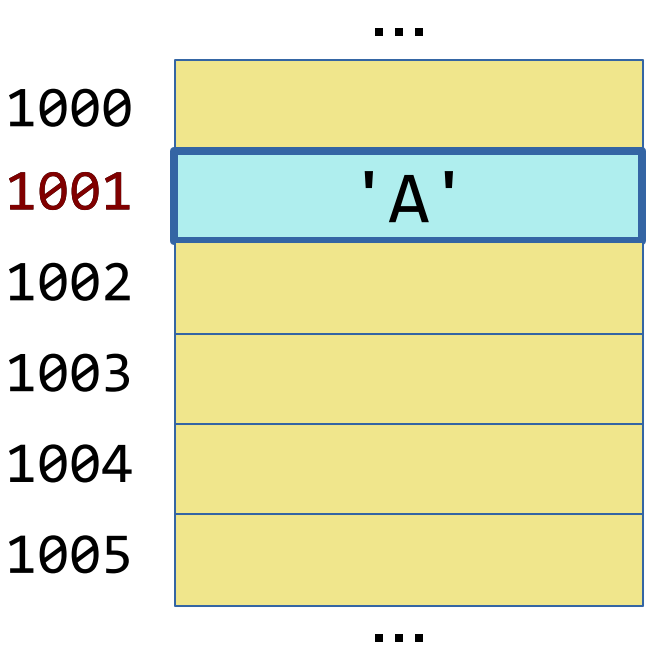
Memory Location and Variables

- A variable declaration allocates memory to store the value of the variable

```
char c = 'A';
```

Memory location 1001 contains value of variable c

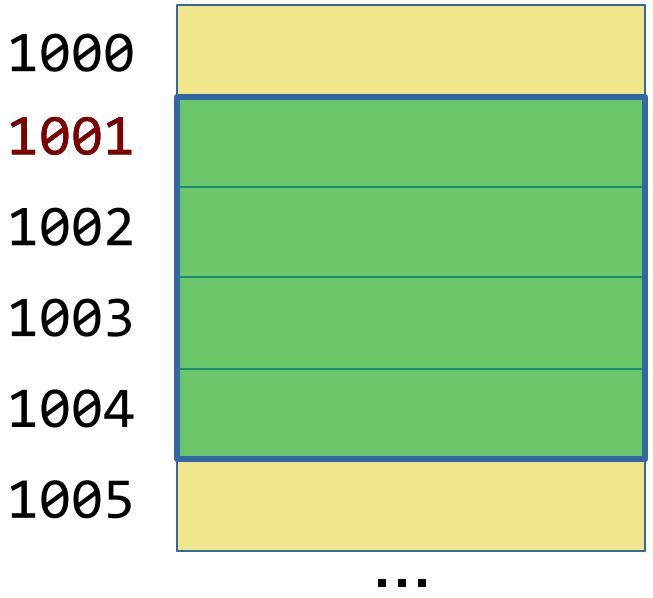
A variable directly references a value



Memory Location and Variables

- In a byte-addressable computer, how do we address a data that occupies more than 1 byte, *e.g.*, int, float or double?

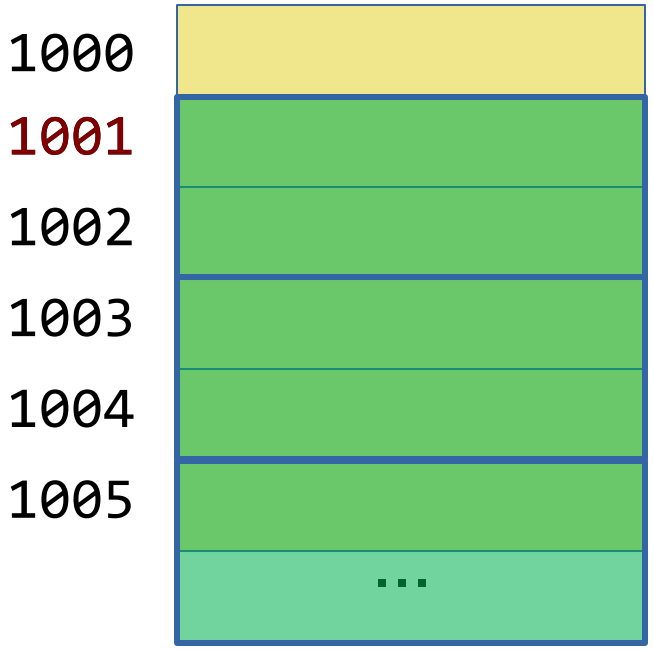
The address of a multi-byte data is the **starting address**



Memory Location and Variables

- In a byte-addressable computer, how do we address arrays?

The address of an array is the **starting address of the first element**



Memory Location and C

- C provides the ability to access specific memory locations, using **pointers**

Pointers are variables that contain memory addresses as their values

Variable vs Pointer

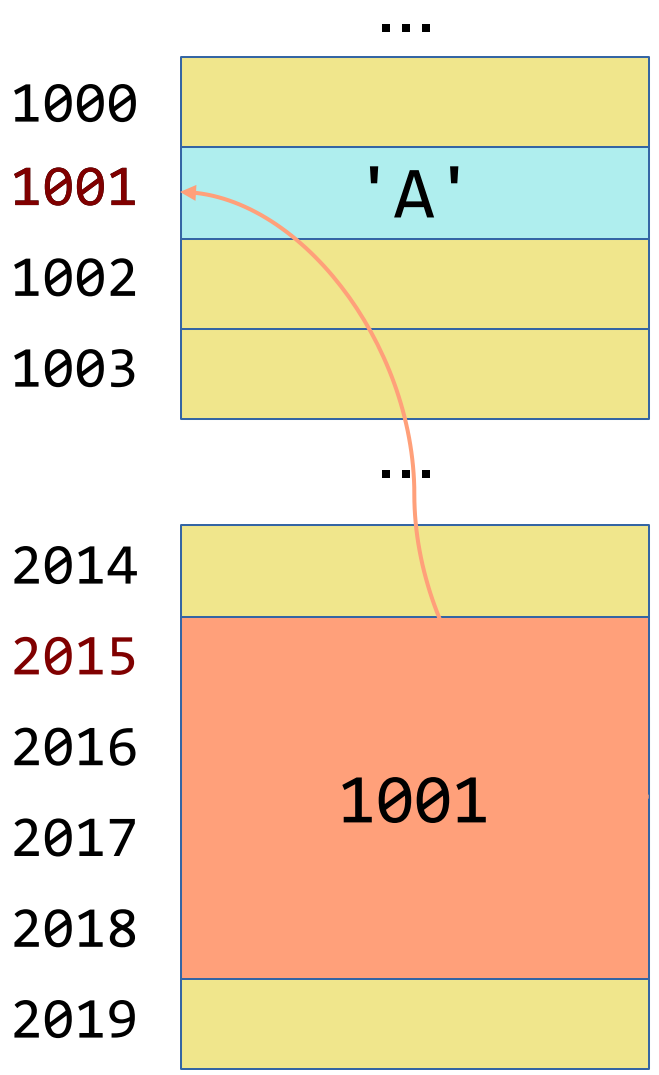
A variable directly references a value

A pointer indirectly references a value

A pointer and a variable

A variable directly references a value

A pointer indirectly references a value



Declaring a Pointer

- Pointers are typed based on the type of entity that they point to
  - To declare a pointer, use * preceding the variable name as in:

```
data_type *name;
```

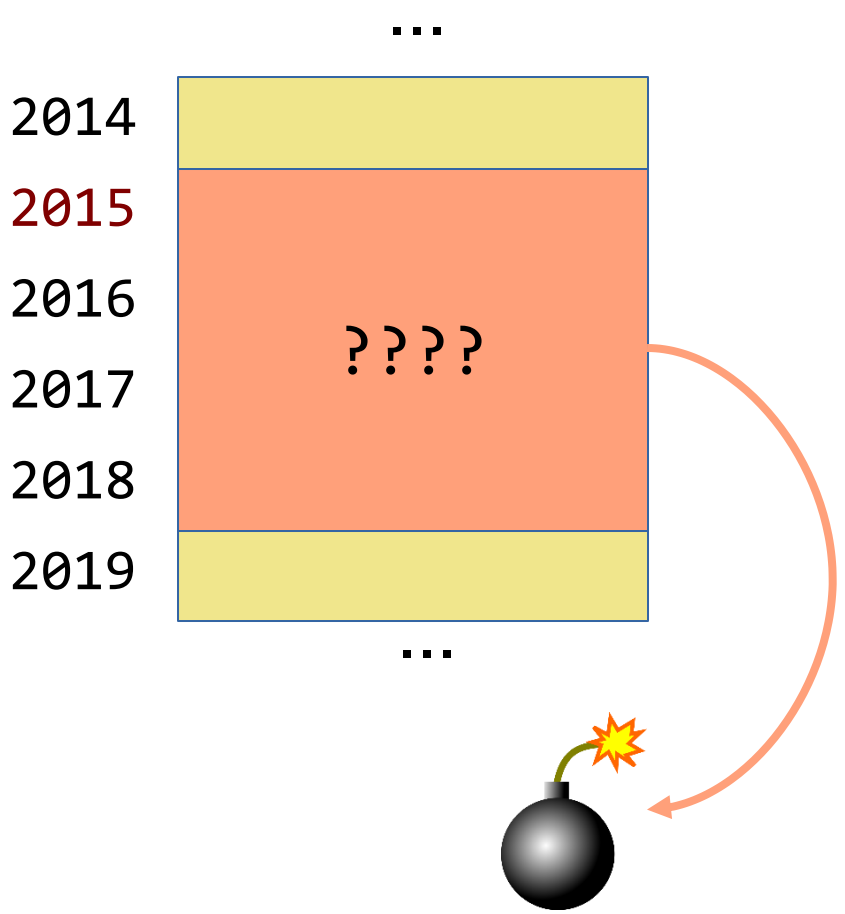
  - Examples:

```
int *p;    // p is a pointer to an int
float *q;  // q is a float pointer
char *r;   // r is a char pointer
int *s[5]; // s is an array of 5 int pointers
```

What Happens in a Pointer Declaration?

```
int *p;
```

- Memory is allocated that can store an address
- The size of this space depends on the number of bits used for addressing
- The initial contents may be some ‘rubbish’ number
  - This means the pointer may point to arbitrary memory locations



Address Operator (&)

- The address (&) operator can be used in front of any variable
  - The operation will return the memory location of the variable

```
&name
```

name can be any ordinary variable or even a pointer variable

- Example:

```
int a, *x;
x = &a;
/* x variable contains address of a, i.e.,
x points to variable a */
```

Indirection Operator (*)

- A pointer variable contains a memory address
- To refer to the *contents* of the variable that the pointer points to, we use indirection operator

```
*name
```

name is a pointer variable

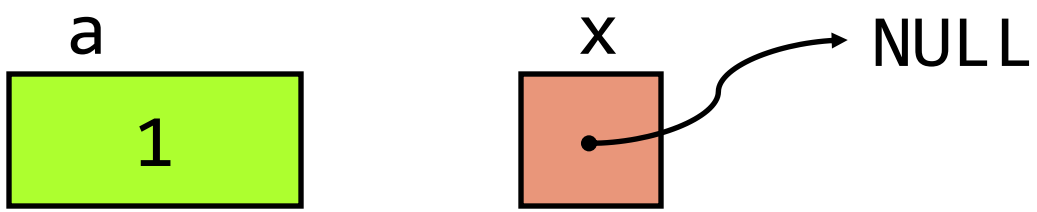
- Example:

```
int a = 100, b, *x;
x = &a;
b = *x;
/* b will be assigned the content pointed
to by x, which is 100 */
```

Graphical Illustration

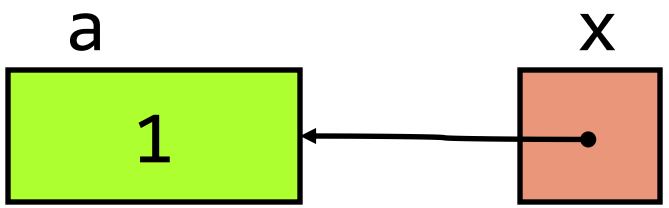
Declaration:

```
int a = 1;
int *x = NULL;
```



`NULL` – pointer literal/constant to non-existent address

Assignment:




```
x = &a;
```

Pointer Basics

- Given:

```
int a = 1, b = 5; int *x;

x = &a;    // What is the value of x ?

*x = *x + 1;      // a = __ ;  b = __ ;

b = *x;
```

- What is the value of b ?

Admin

- Assignment 2 has been released
  - Due date is 22 October, 7 pm

Usage of Pointers

1) Provide an alternative means of accessing information stored in arrays

2) Provide an alternative (and more efficient) means of passing parameters to functions

3) Enable dynamic data structures, that are built up from blocks of memory allocated from the heap at run time

Pointers and Arrays (1)

- **Arrays in C are pointed to**, i.e. the variable that you declare for the array is actually a **fixed pointer** to the first array element

- Example:

```
int z[10] = {1, 2, 3};
```

  - `z` is a fixed pointer, it points to the address of the first element `z[0]`
  - In other words, `z == &z[0]`

Pointers and Arrays (2)

- Array elements are usually accessed using `[ ]` (with the index)

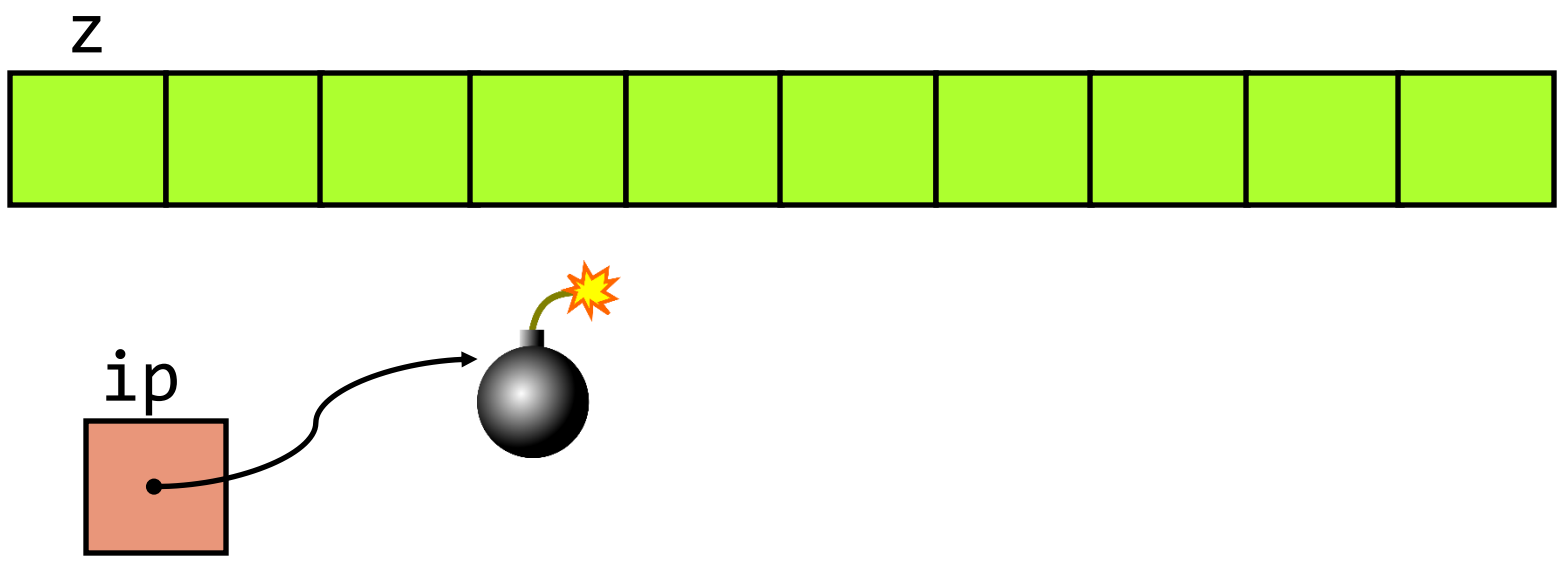
- Pointers can also be used to access array elements

```
int z[10], *ip;
ip = &z[0];
```

  - `z[0]`, `ip[0]`, `*z`, or `*ip` can all be used to access the *first* element of the array `z`

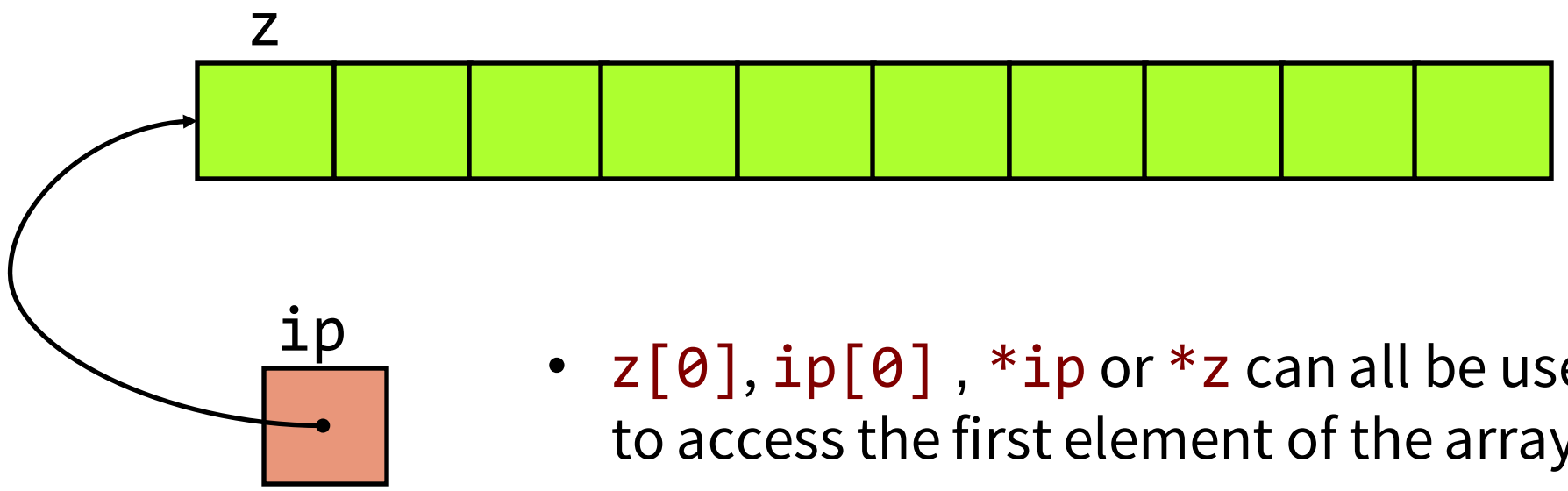
Graphical Illustration

```
int z[10], *ip;
ip = &z[0];
```



Graphical Illustration

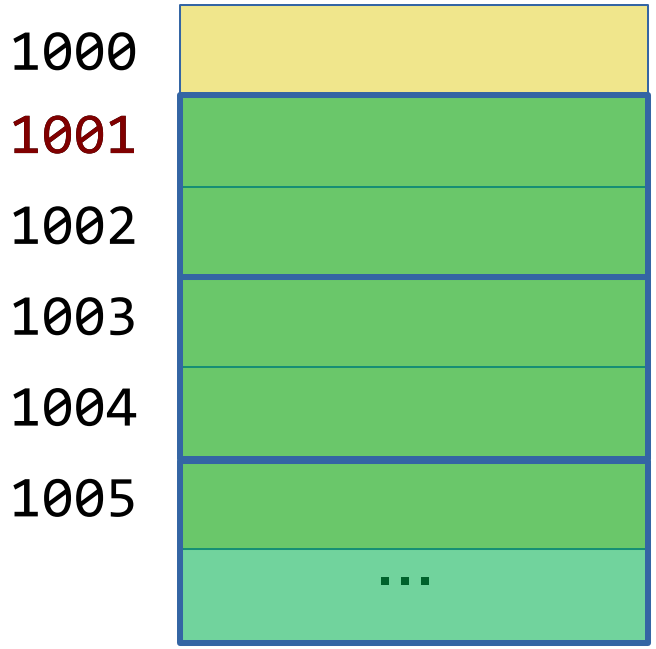
```
int z[10], *ip;
ip = &z[0];
```



- `z[0]`, `ip[0]` , `*ip` or `*z` can all be used to access the first element of the array `z`

How To Access Next Element Using Pointer?

- What about accessing `z[1]` using pointers ?



Is it `*(ip+1)`?

- Hmmm…
- Since `ip` is an address, adding `1` will just point to the next byte
- But since the array consists of ints (which are more than 1 byte), `ip+1` will still point to a certain part of the first element?

Pointer Arithmetic

- Addition and subtraction can be performed on pointers
- Suppose :

```
data_type *name;
```

`name + k`	Evaluated as `name + k*sizeof(data_type)`
`name - k`	Evaluated as `name - k*sizeof(data_type)`

Pointers and Arrays (3)

- **Arrays in C are pointed to**, i.e. the variable that you declare for the array is actually a **fixed pointer** to the first array element

- Example:

```
int z[10] = {1, 2, 3};
```

  - `z` is a fixed pointer, it points to the address of the first element `z[0]`
  - In other words, `z == &z[0]`
  - In general, `z+i == &z[i]`

Pointers and Arrays (4)

- Array elements are usually accessed using `[ ]` (with the index)

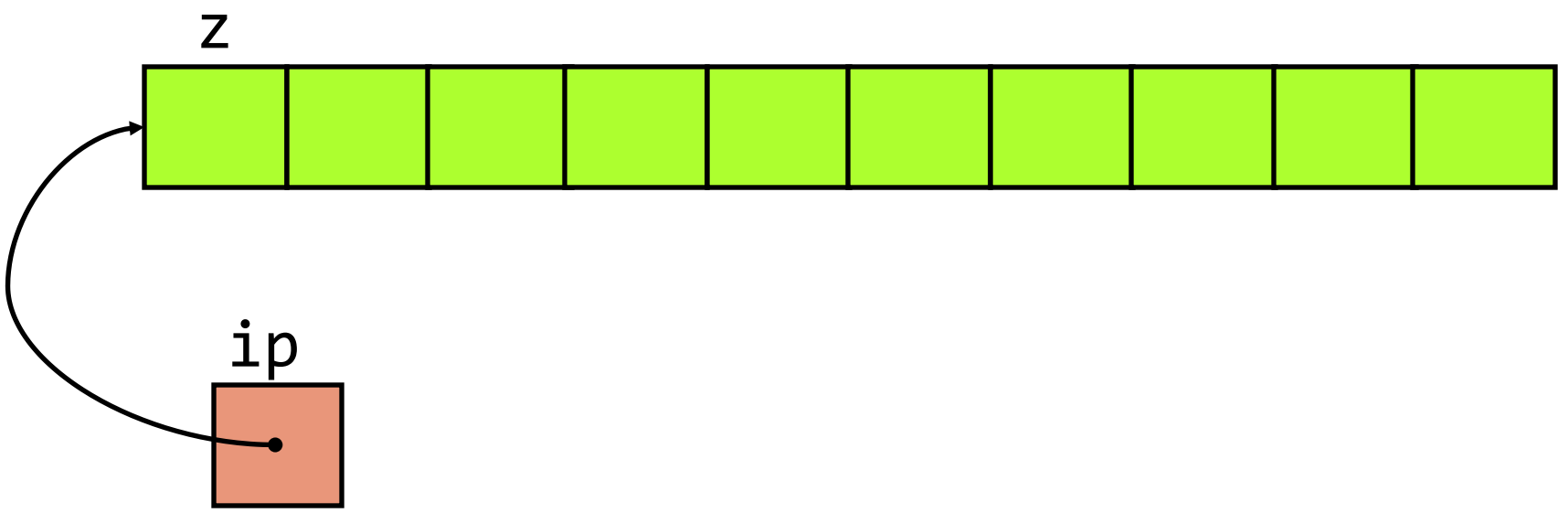
- Pointers can also be used to access array elements

```
int z[10], *ip;
ip = &z[0];
```

  - `z[i]`, `ip[i]`, `*(z+i)`, or `*(ip+i)` can all be used to access the `i`*th* element of the array `z`

Graphical Illustration

```
int z[10], *ip;
ip = &z[0];
ip++; // ip = ip + 1
```



Graphical Illustration

```
int z[10], *ip;
ip = &z[0];
ip++; // ip = ip + 1
```