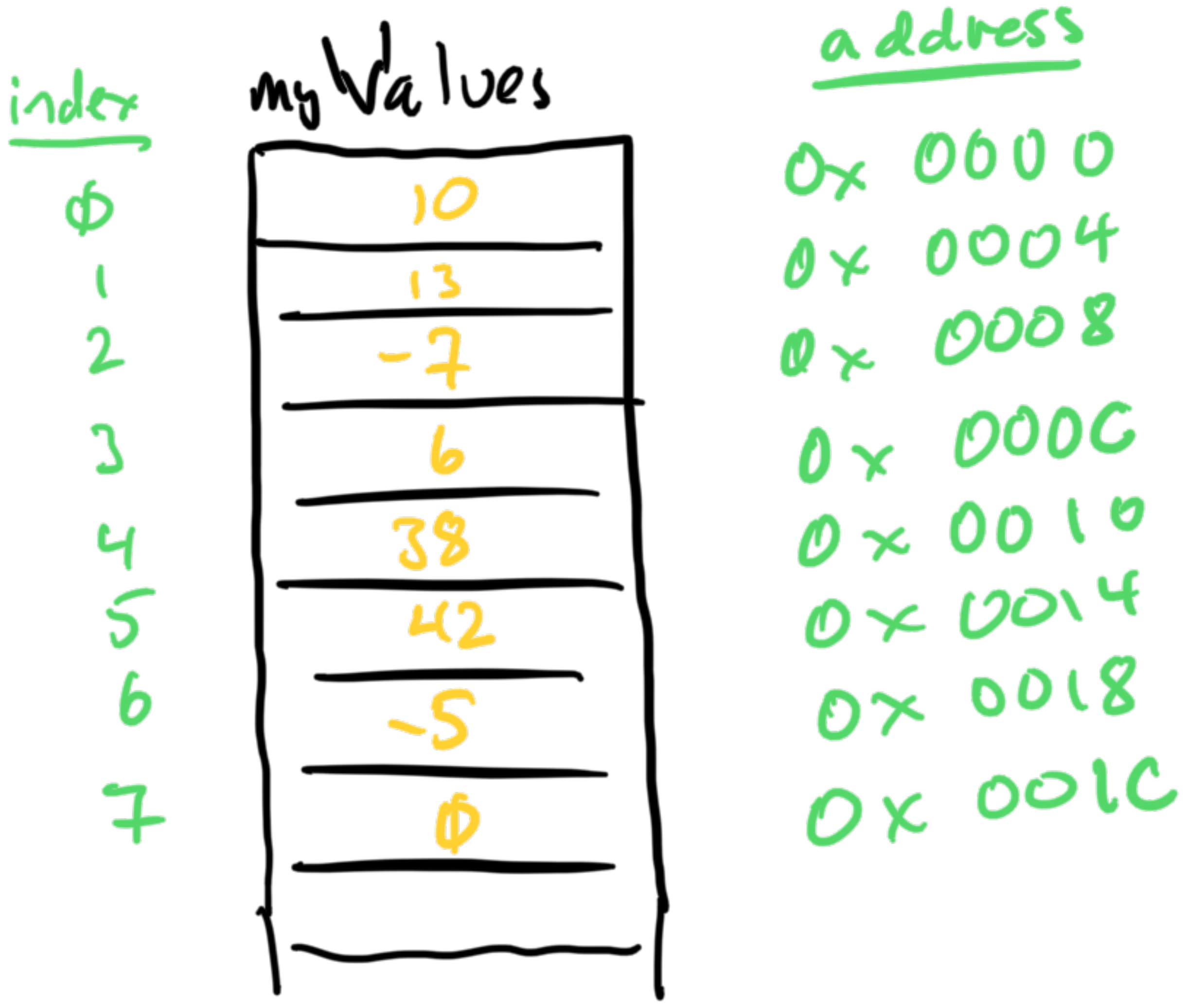
Pointers in C

Consider, for a moment, the following array of ints:



index	myValues	address
Ø	10	0x 0000
1	13	0x 0004
2	-7	0x 0008
3	6	0x 000C
4	38	0x 0010
5	42	0x 0014
6	-5	0x 0018
7	Ø	0x 001C

```
int myValues[8];
```

Suppose, now, we are interested in the <u>value</u> stored in the <u>fifth</u> position (38).

```
i = myValues[4]
```

We also know that the <u>address</u> of the first element of the array is important. This address is referred to as a *pointer*.

```
printf("%p\n", myValues);
```

or

```
printf("%p\n", &myValues[0]);
```

↑ "the address of"

Q1: What if we wanted to *store* this address, for later use?

```
int*  p  =  &myValues[0];
```

↖ "the address of"

$\underbrace{\texttt{int*}}_{\text{pointer to an integer}}$

English: get the address of the 0-th element of the myValues array, and store that address in the int-pointer variable, p.

NOTE !!!

- Some text books, professors, and other evil beings write this as:

```
int    *p   =  . . . . .
```

↑ star next to variable !!

- I think this is TERRIBLE!
- much better to think of

  "int*" as a new variable type

Q2: What if we wanted to know the value stored at a particular address ???

-> we need a new operator for this

-> something that means "the value stored at the address ..."

```
int*  p  =  &myValues[Ø];

int  i  =  *p;
```

-> this is called the pointer dereferencing operator.

-> it only works on pointers.

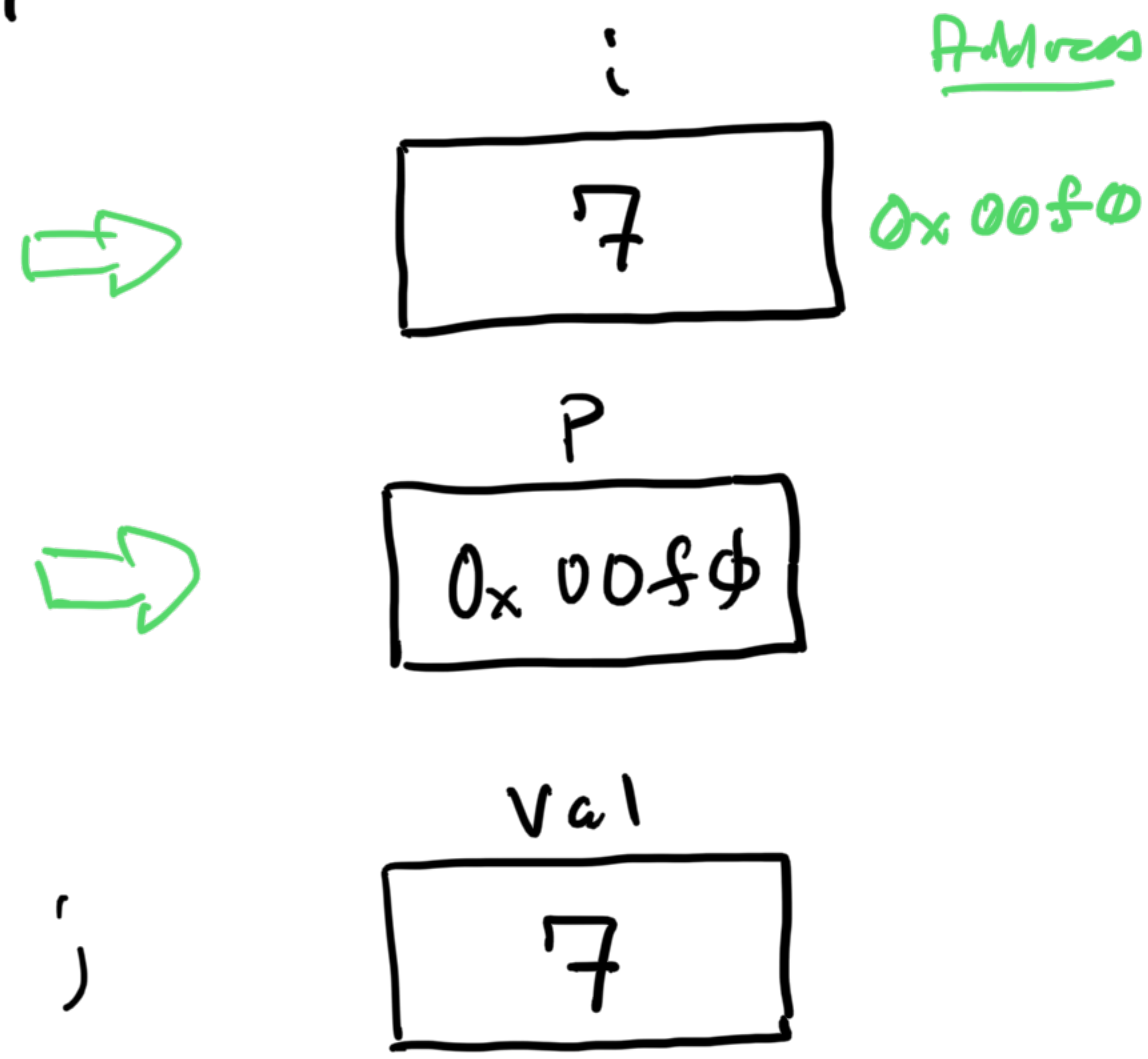
-> it's unfortunate they chose "*" ☹.

Let's look at the project called Basic Pointers ... lot's of examples here, which illustrate these concepts, and more.

```
int i = 7;

int* p = &i;

int val = *p;
```



Q3: What happens if I now change the value of i ???

Does val change?

Answer: NO. p is "linked" to i. But, val is not "linked" to p ... the dereferencing operator just "evaluates" a single time.

---

the function!! This

Using pointers with "functions"..

Is where the real power comes in ☺

(Note: C already does this automagically for arrays!)

<u>Task</u>: Write a function that returns the n-th character of a string (i.e. char array)

<u>Method 1</u>: Pass a <u>copy</u> of the entire char array to the function.

```
Char getChar(char s[], int n){
    char thisChar = s[n-1];
    return thisChar;

}
```

↑ return the n-th character

↑ copy of char array (or is it?)

↑ copy of # of character to find.

Method 2 :

Pass a pointer to the first character of the string.

```
char    getCharPointer(
                char*  s, int n){

        char*  pa   = &s[n-1];
        char  thisChar = *pa;
        return thisChar;

}
```

Just to be clear, there is actually no difference between ① and ② in modern C implementations! But, we should use method 2, in

we should use ,
general, as a matter of readability
and clarity!! That way, the reader
understands one is passing a pointer.

Fibonacci Sequence

0	1	2	3	4	5	6	7	8	....		
↓	↓	↓	↓	↓	↓	↓	↓	↓			
0,	1,	1,	2,	3,	5,	8,	13,	21,	34,	55,	.....

Question: What is the n^{th} Fibonacci number?

Method 1: (i) initialize 0, 1
(ii) loop and add

Method 2:

(i) $$f_n = \frac{\phi^n - (-\phi)^{-n}}{\sqrt{5}}$$

where $\phi = \frac{1+\sqrt{5}}{}$ ≈ 1.61803

2

Example : $$f_6 = \frac{\phi^6 - (-\phi)^{-6}}{\sqrt{5}}$$

$$\approx \frac{17.94427 - (0.05572809)}{2.236067}$$

$$= 8.000003$$

N.B. We have to be careful when dealing with large numbers !!

What is $f(100)$?