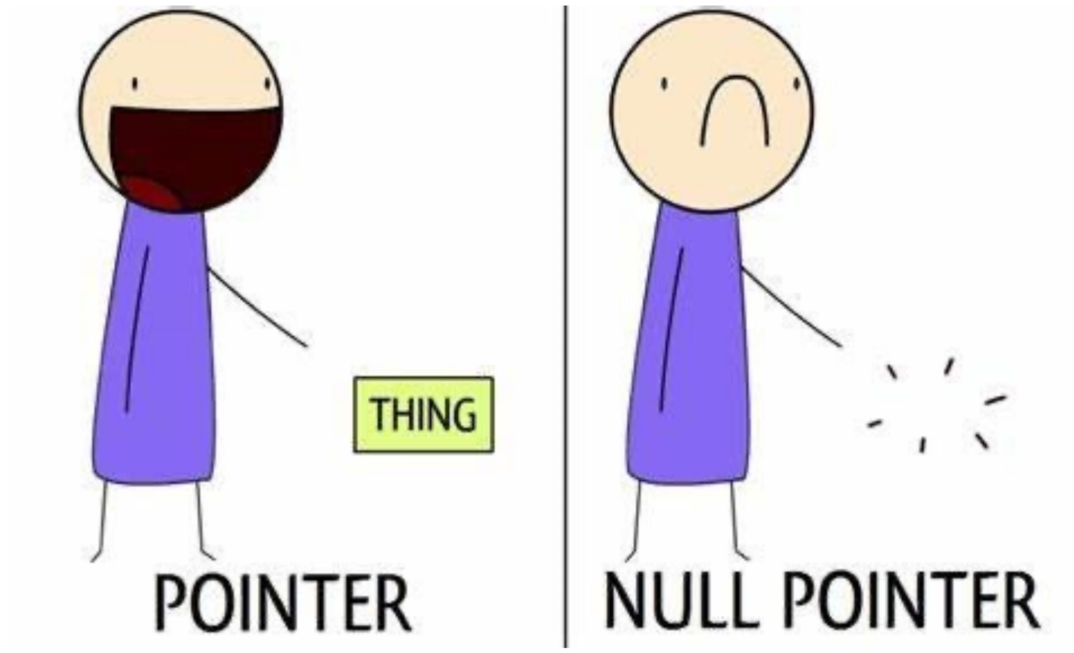
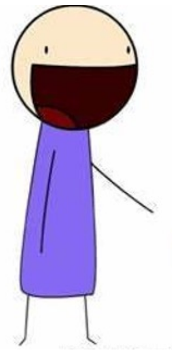
ALGORITHM AND COMPUTATIONAL THINKING 2

WEEK 6 – Pointers



🥇 Session objectives 🥇

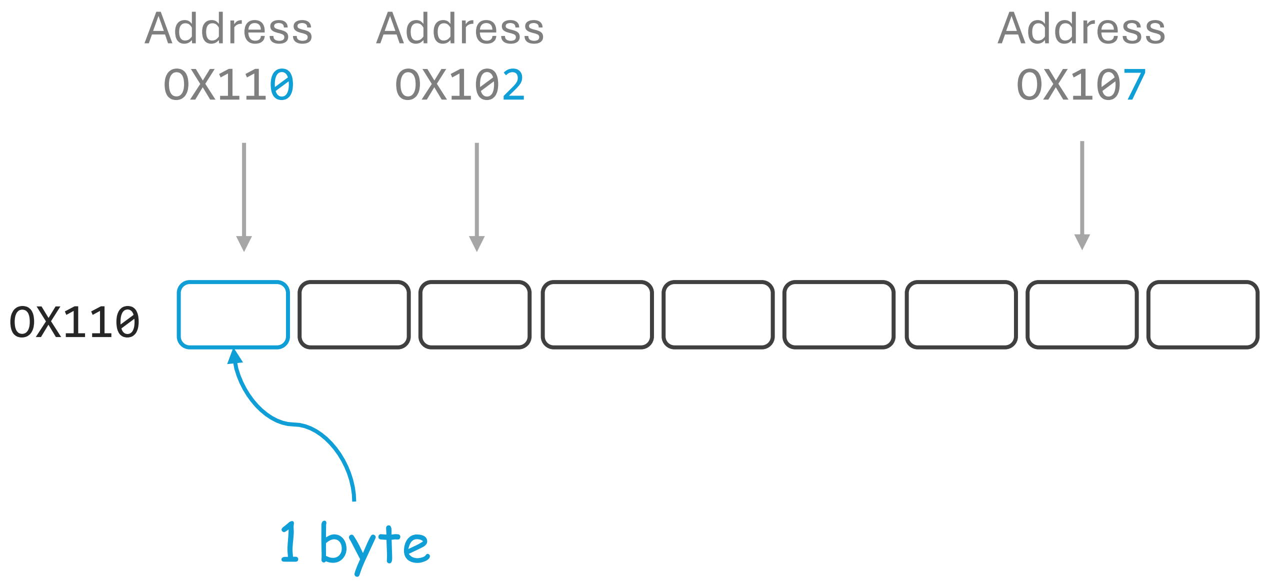
- ✓ Understand **memory addresses** and **pointers**.
- ✓ Declare and initialize **pointers**.
- ✓ Use the **address (&)** and **indirection (*)** pointer operators.
- ✓ Use **pointer arithmetic** to iterate through arrays.
- ✓ Use pointers to pass arguments to functions **by reference**.



EXPLAIN

Memory is structured in **bytes**

In a computer, each memory address stores **1 byte (8 bits)**



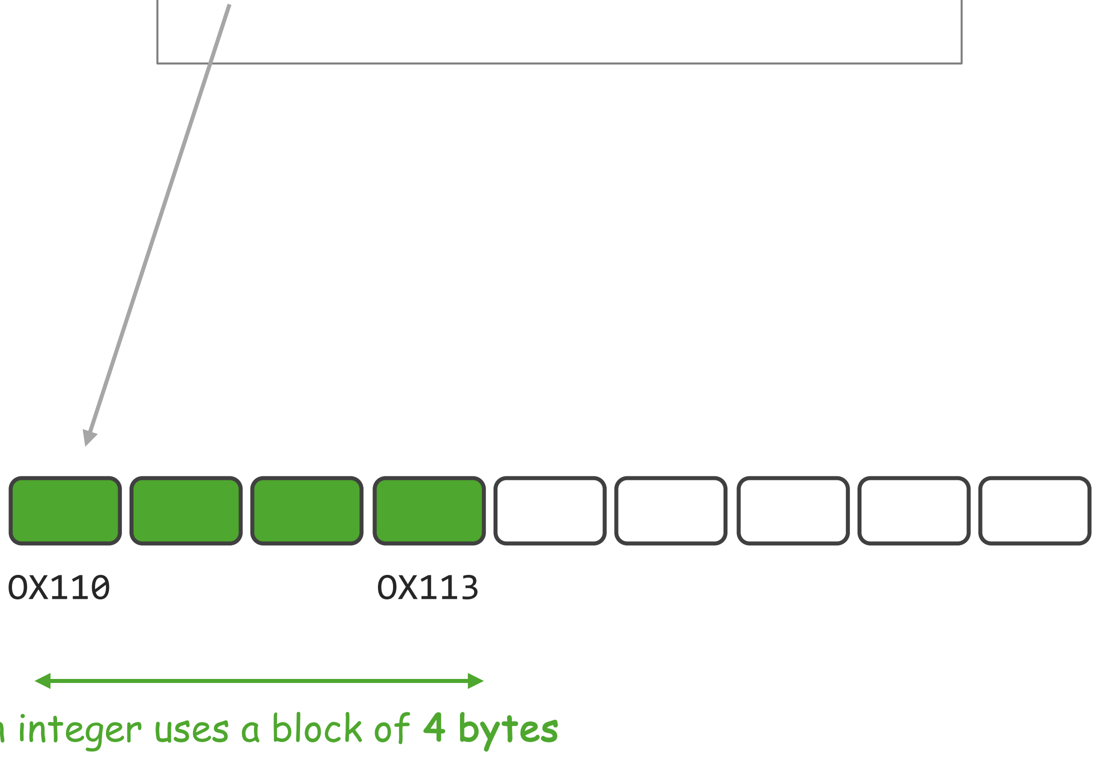
Simplified representation of the memory structure

EXPLAIN

Memory **allocation**

When a variable is created in C, a **block of memory** is reserved.

```
// Create an integer
int score = 25;
```

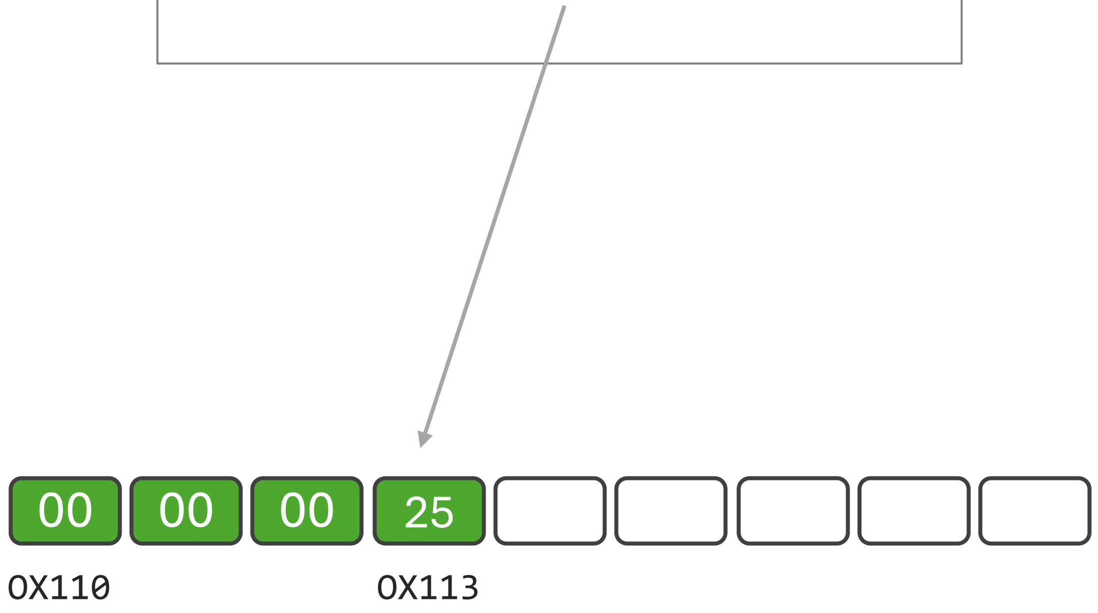


EXPLAIN

Memory **allocation**

The variable value can **then be stored** in this block of memory.

```
// Create an integer
int score = 25;
```

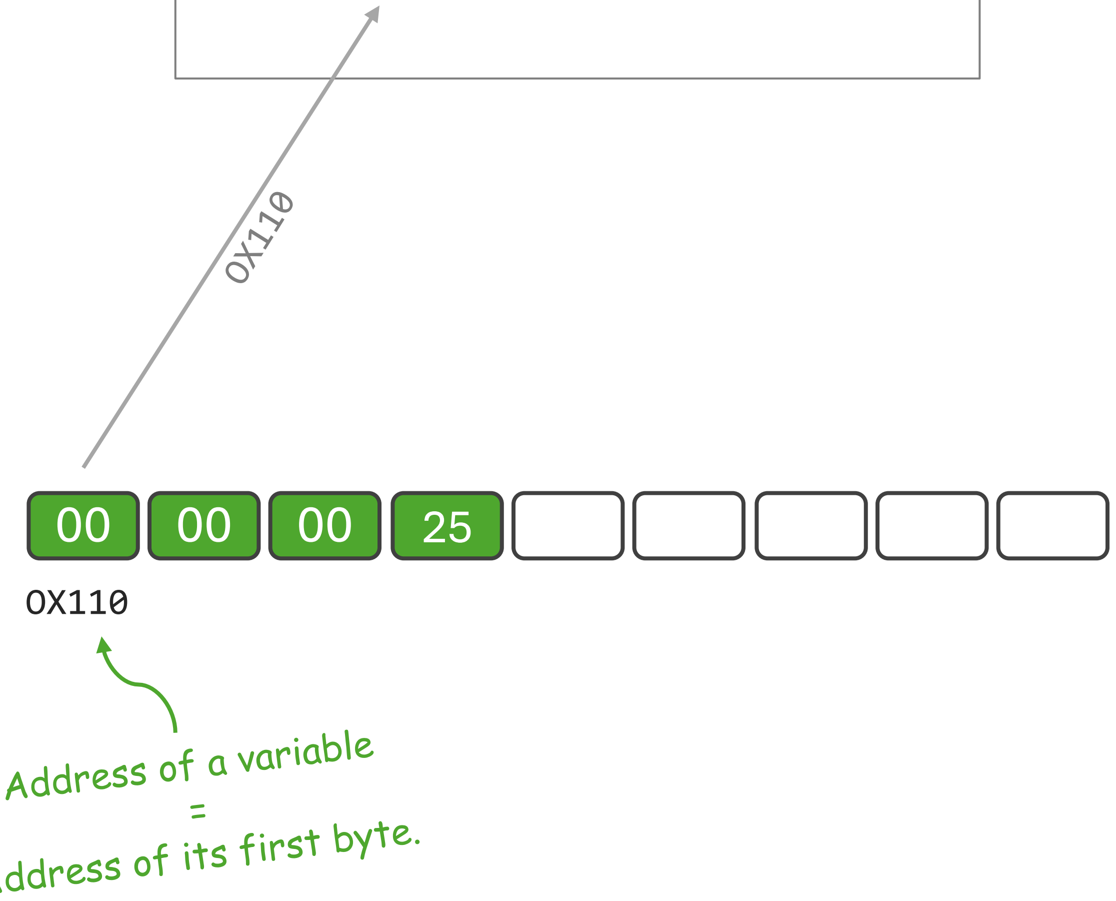


EXPLAIN

Memory **allocation**

The **address of the variable** in the memory is kept for future access.

```
// Create an integer
int score = 25;
```

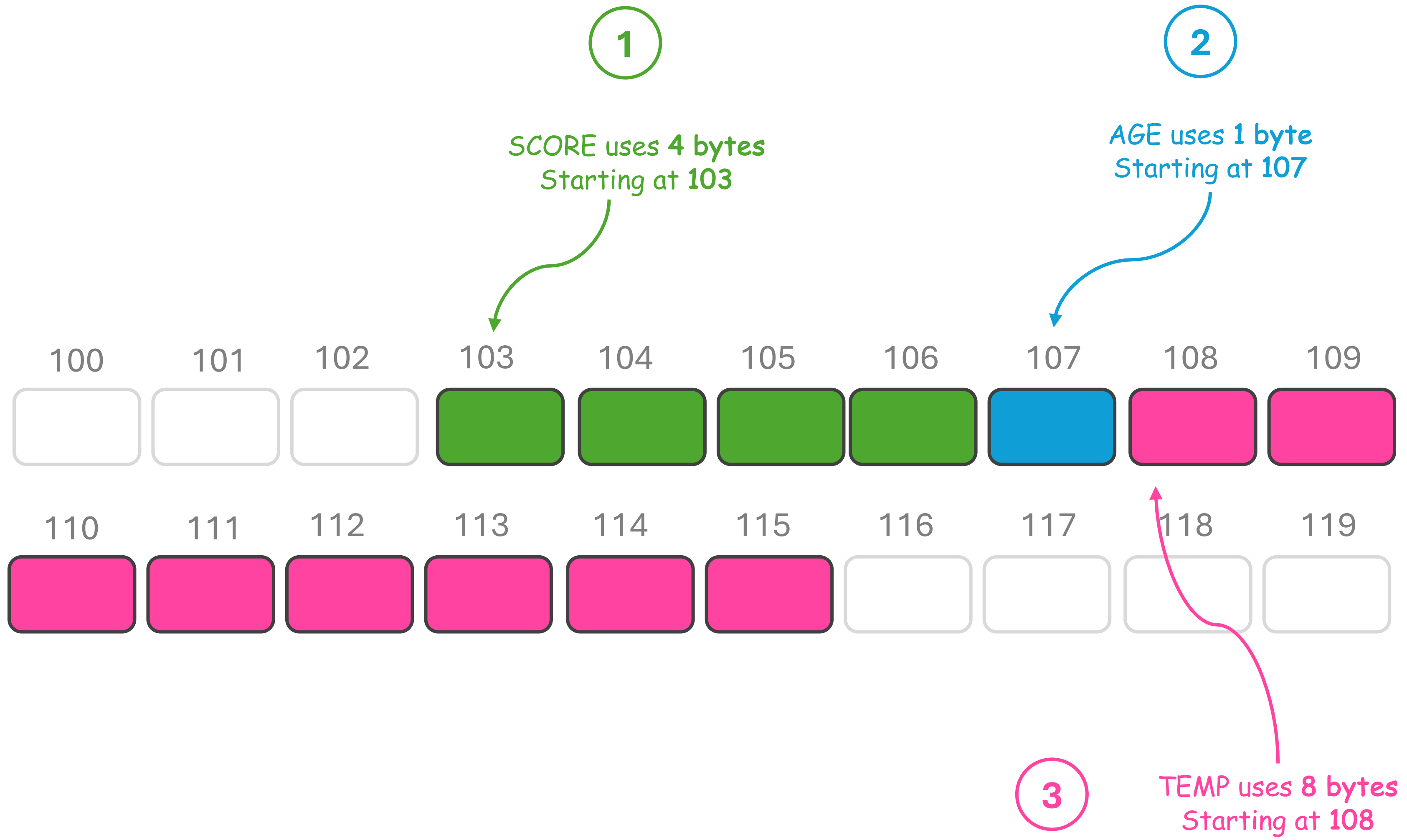


EXPLAIN

Memory is structured in **bytes**

Variables are stored in **aligned**, **contiguous** memory blocks as the program runs.

```
int SCORE = 25;

char AGE = 'A';

double TEMP = 37.8;
```



EXPLAIN

MORE INFO

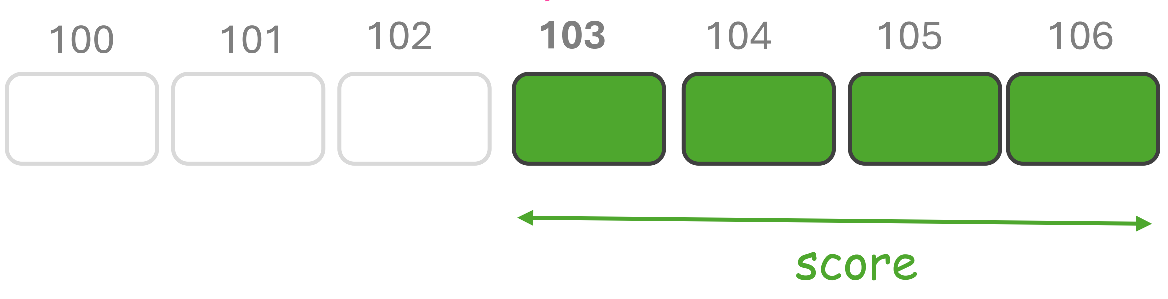
How to access the memory?

To access to the **hexadecimal address** of a variable, we use the **reference operator** (&)

```
int score = 25;

// Print the address of the variable
printf("%p", &score);
```

P is the formatter
For hexadecimal addresses

```
0x103
```



⚠ The **hexadecimal address** depends where the variable is stored on the computer.

REFLECT

Why do we place a reference operator (&) in scanf() ?

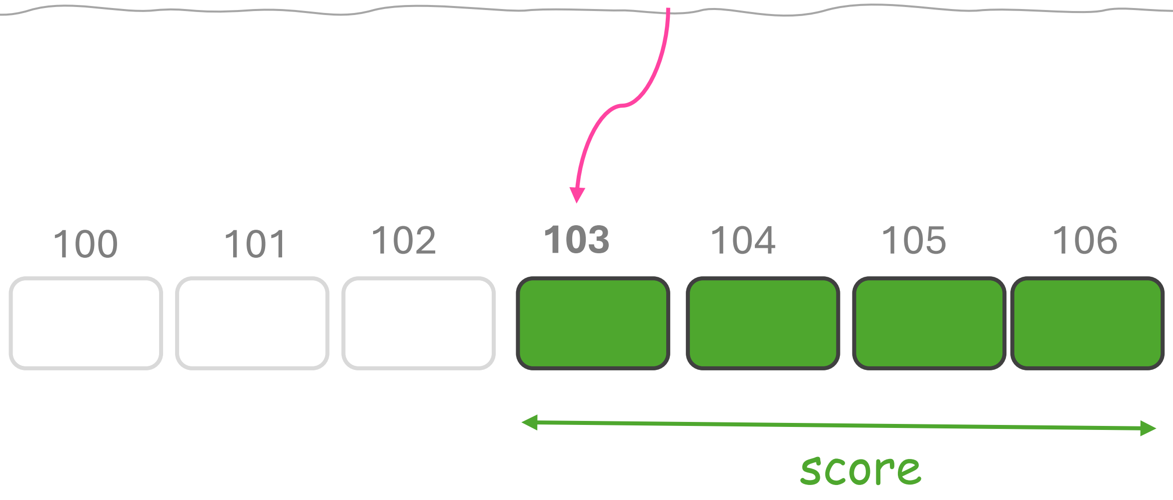
```
int score;
scanf("%d", &score);
```

ANSWER

Why do we place a reference operator (&) in scanf() ?

To give scanf() the **memory address** where to write the scanned value.

```
int score;
scanf("%d", &score);
```

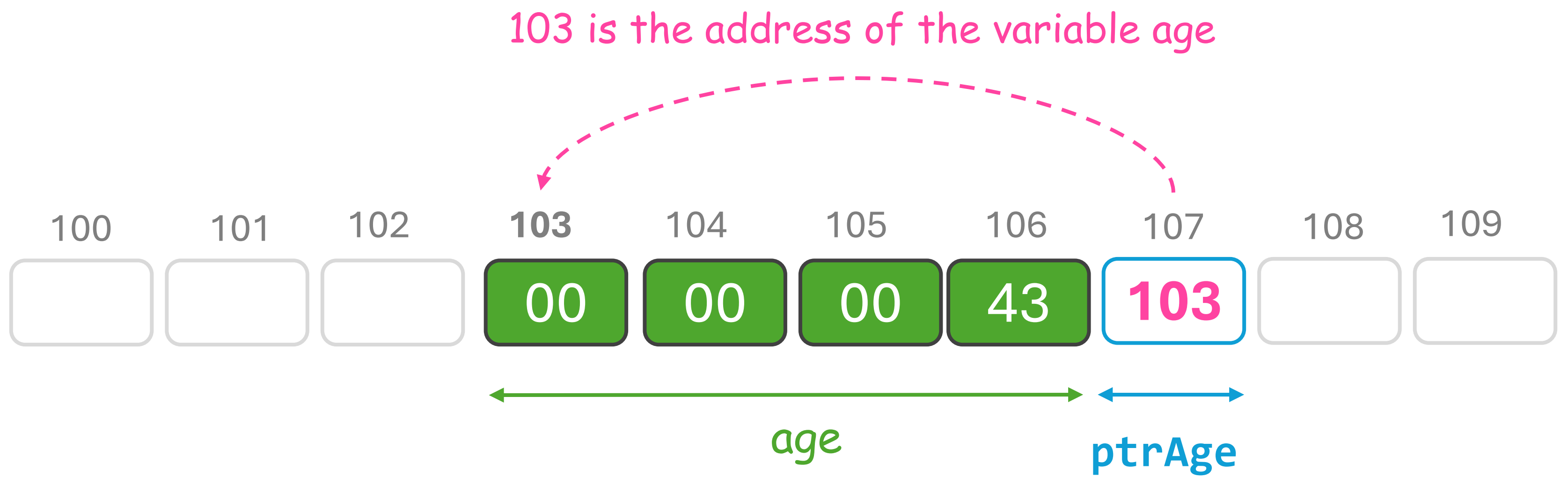


EXPLAIN

What is a pointer ?

A **pointer** is a variable that stores the **memory address** of another variable.

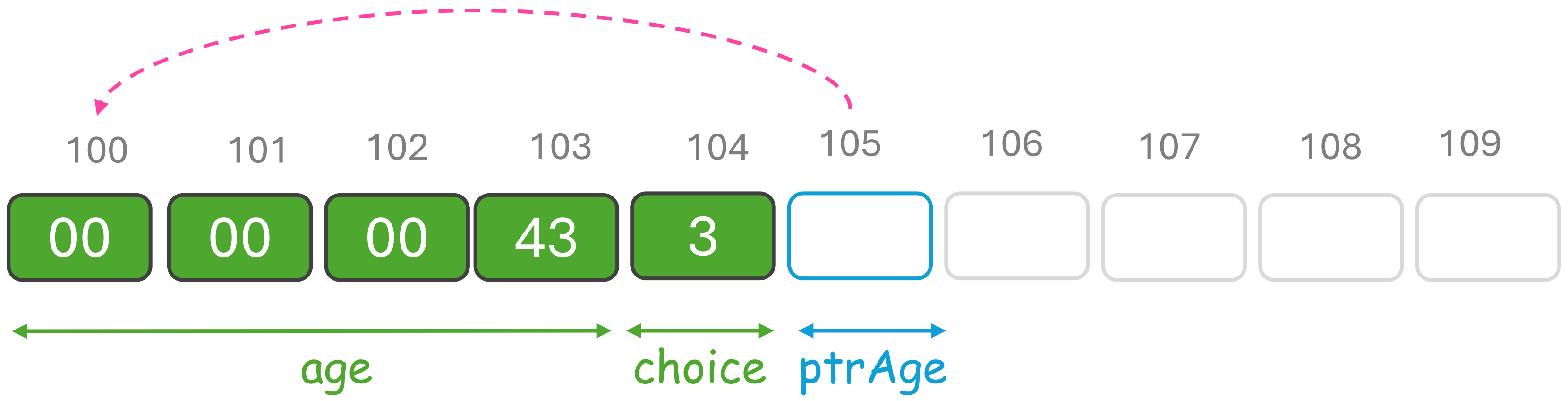
```
int age = 43;

// Pointer storing the address of age
int* ptrAge = &age;
```



QUIZ

What will this code print?

```
int age = 43;
char choice = 3;

int* ptrAge = &age;
printf("%p", ptrAge);
```

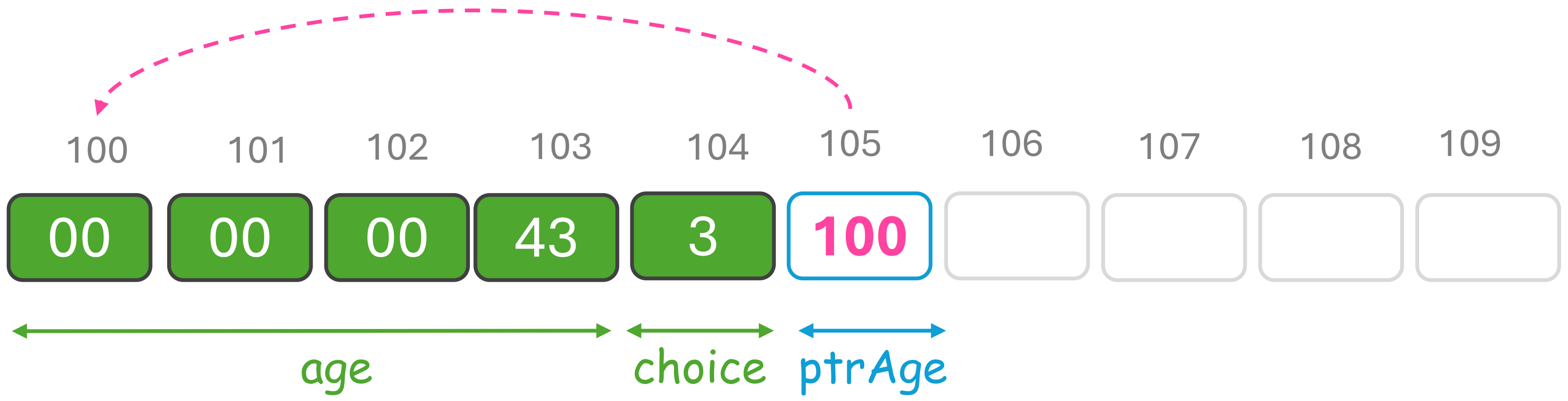


A	00	B	101	C	43	D	100

What will this code print?

```
int age = 43;
char choice = 3;

int* ptrAge = &age;
printf("%p", ptrAge);
```

The pointer refers too
The address of the
variable age.



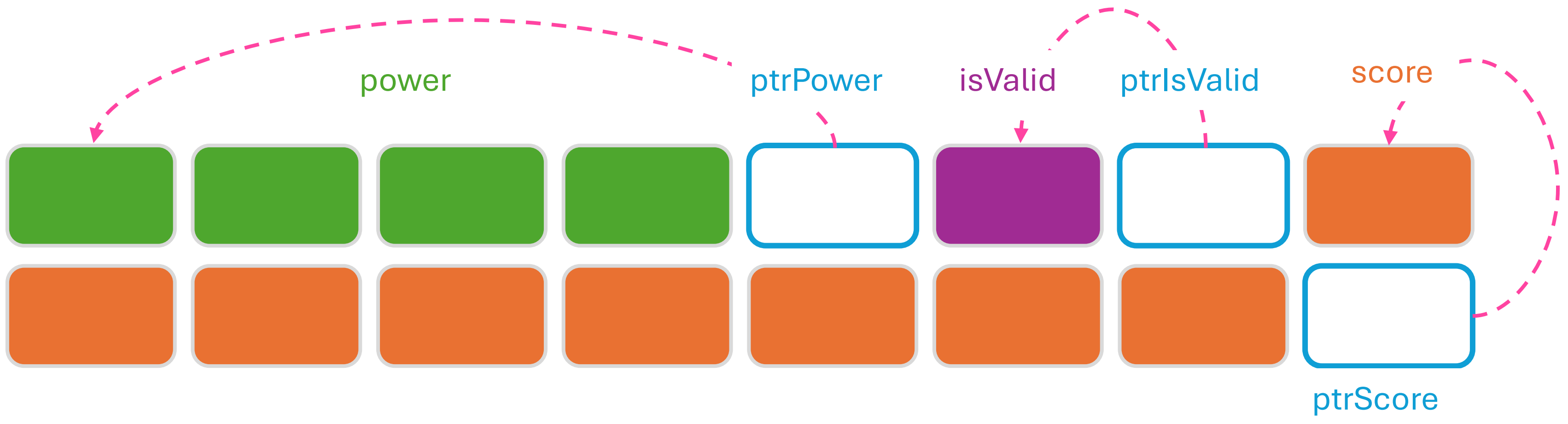
A 00 B 101 C 43 [✓ D] 100

EXPLAIN

We can point to anything

```
int power;
int* ptrPower = &power;                    //  A pointer to integer

bool isValid;
bool* ptrIsValid = &isValid;               //  A pointer to boolean

double score;
double* ptrScore = &score;                 //  A pointer to double
```

Use ptr as a prefix on your pointer variables.



EXPLAIN

Access to **the value** referenced by a pointer

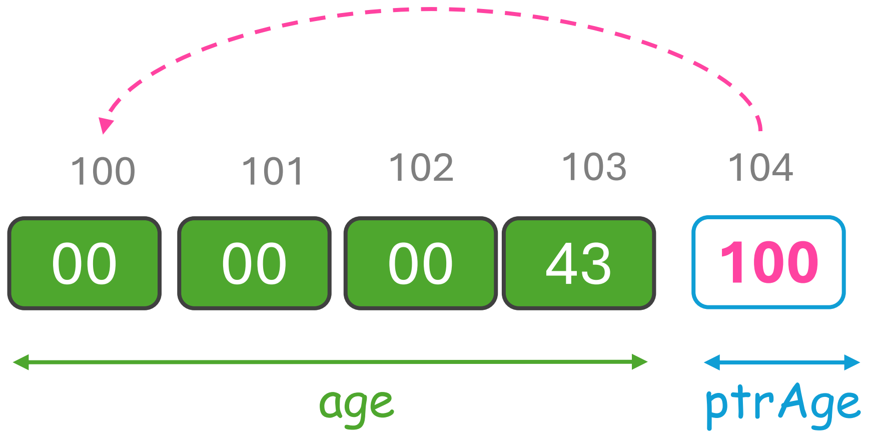
To access to the value referenced by the pointer, we use the **dereference operator (*)** *(i.e the 'value of')*

```
int age = 43;
int* ptrAge;
ptrAge = &age;

int value = *ptrAge;
```

100 (address of age)

43 (value referenced by the address)

Dereference operator



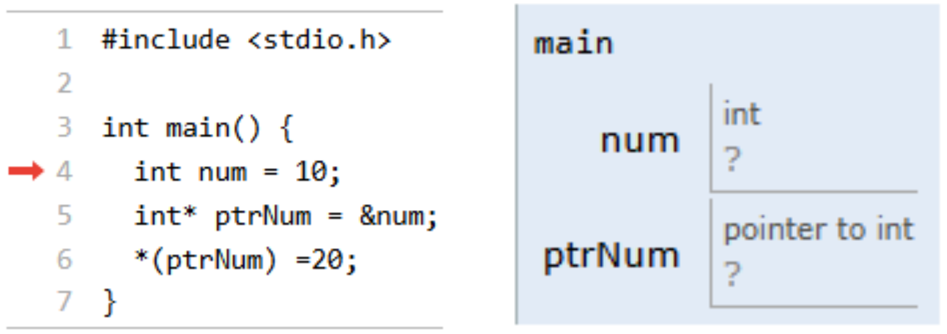
REFLECT

Change the value referenced by a pointer

Q1 – Analyze the below code

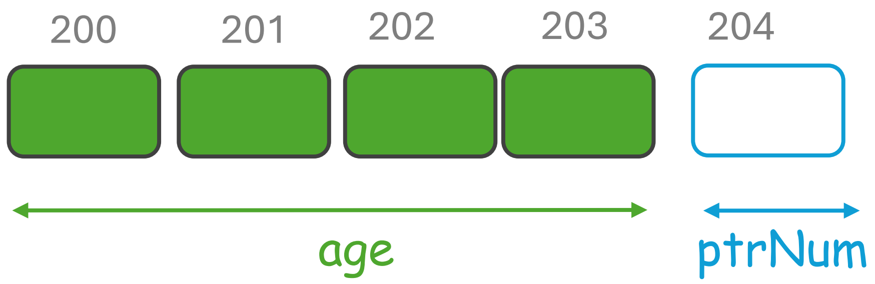
```
int num = 10;
int* ptrNum = &num;
*ptrNum =20;
```

Q2 – Run the code **step by step** - online debugger



```
#include <stdio.h>

int main() {
  int num = 10;
  int* ptrNum = &num;
  *(ptrNum) =20;
}
```

Q3 – After understanding the code behavior, write the **memory status** at the end of the code:



ANSWER

Change the value referenced by a pointer

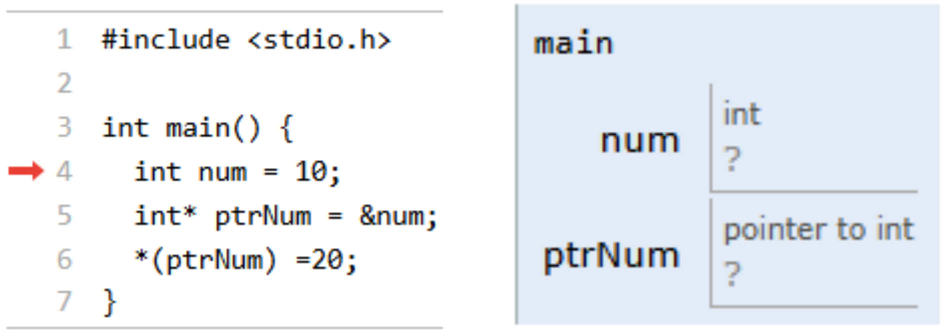
Q1 – Analyze the below code

```
int num = 10;
int* ptrNum = &num;
*ptrNum =20;
```

We define a pointer
That refers to the int num.

We modify the value referenced by the pointer.

Q2 – Run the code **step by step** - online debugger

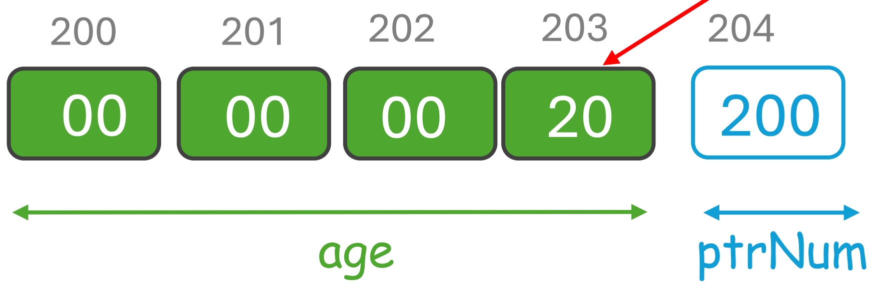


```
#include <stdio.h>

int main() {
  int num = 10;
  int* ptrNum = &num;
  *(ptrNum) =20;
}
```

main	
num	int ?
ptrNum	pointer to int ?

Q3 – After understanding the code behavior, write the **memory status** at the end of the code:

We modify the value referenced by the pointer.



EXPLAIN

⚠ 2 usages of the same sign *

To **create** a **pointer** variable

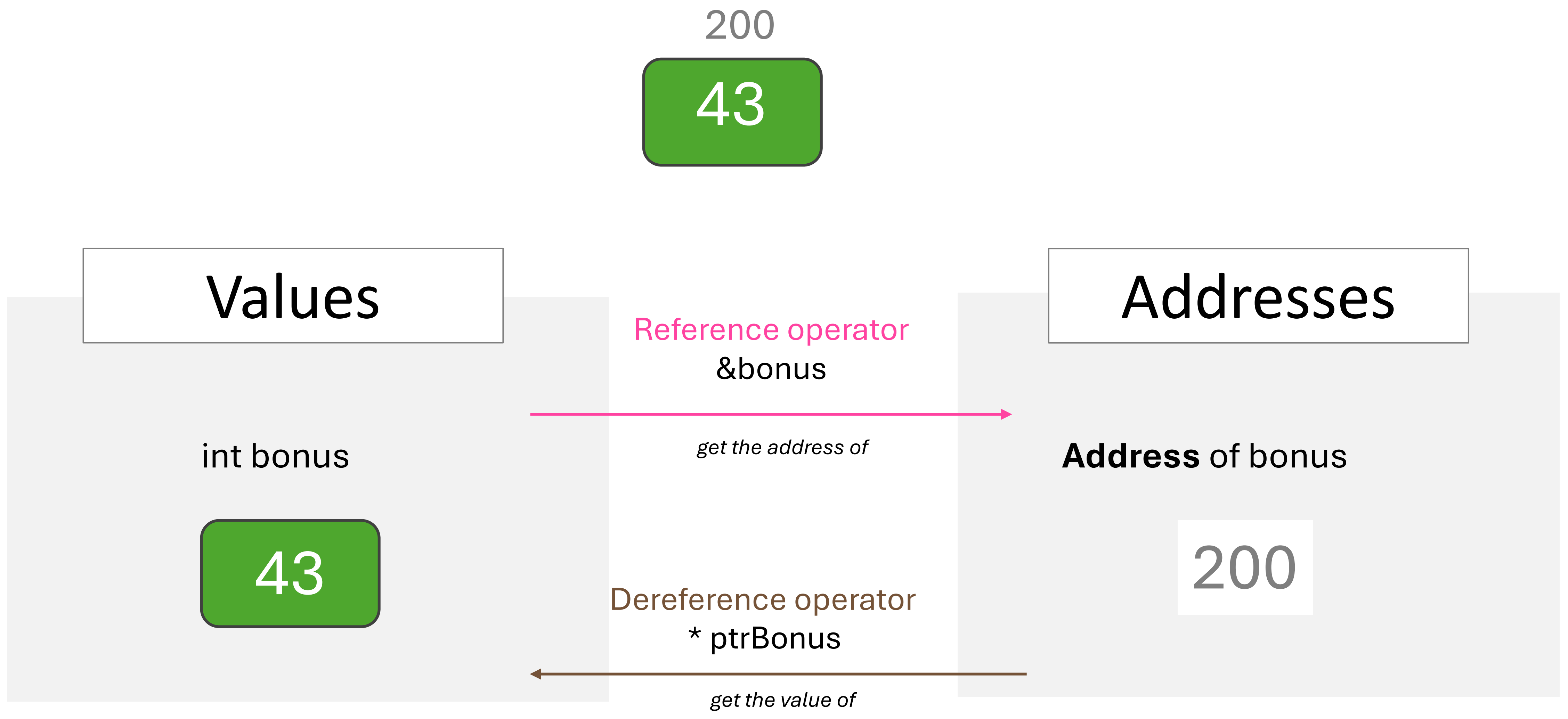
```
int* ptrAge = NULL;
ptrAge = &age;
```

To **access to the value**
(*dereference operator*)

```
*ptrNum = 20;
printf("%d", *ptrNum);
```

EXPLAIN

Reference VS dereference operators



QUIZ

What will this code display?

```
int a    = 10;
int* p   = &a;
*p       = *p + 5;

printf("%d", a);
```

A 10

B 5

C 15

D Compilation error

What will this code display?

```
int a    = 10;
int* p   = &a;
*p       = *p + 5;

printf("%d", a);
```

P is a pointer on a

Increment the value of the reference of 5

The value a is now 15

A 10

B 5

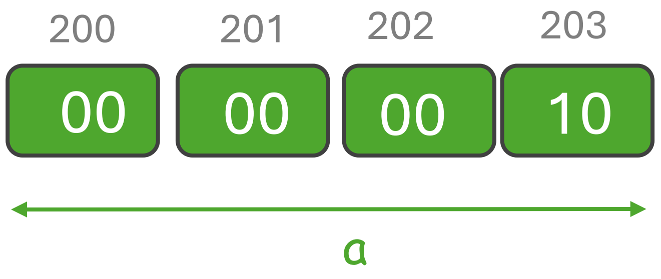
[✓ C] 15

D Compilation error

QUIZ

What will this code display?

```
int a = 10;
a = a + 10;

printf("%p", &a);
```



A 10

B 20

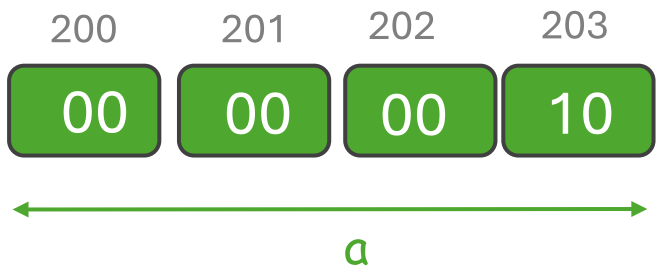
C 0x200

D 0x203

ANSWER

What will this code display?

```
int a = 10;
a = a + 10;

printf("%p", &a);
```



A 10

B 20

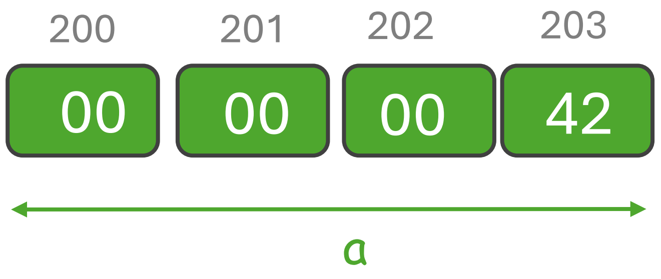
[✓ C] 0x200

D 0x203

QUIZ

What will this code display?

```
int x = 42;
printf("%d",  *(&x) );
```



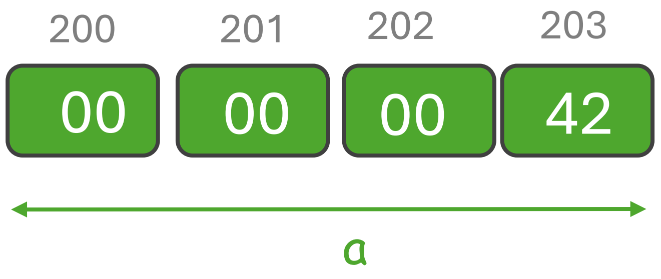
A 42

B 0x200

C Error: invalid syntax

What will this code display?

```
int x = 42;
printf("%d",  *(&x) );
```



*(&X) = X

[✓ A] 42

B 0x200

C Error: invalid syntax

QUIZ

What will this code display?

```
int score = 15;
int* ptrScore = &score;
*ptrScore = 30.5;

printf("value = %d", score );
```

A. 15
B. 30.5
C. 30
D. Compilation error or undefined behavior15

ANSWER

What will this code display?

```
int score = 15;
int* ptrScore = &score;
*ptrScore = 30.5;

printf("value = %d", score );
```

A. 15

B. 30.5

[✓ C.] 30

D. Compilation error or undefined behavior15

- This assigns 30.5 to score
- But the pointer **refers** to a int, **not a double**
- So the value 30.5 (a double) gets implicitly **converted** to an int.

QUIZ

Fill out the **gap**

```
int a = 10;
int* ptr = &a;

___ b = *ptr;
```

A. int

B. int*

C. Compilation error

ANSWER

Fill out the **gap**

```
int a = 10;
int* ptr = &a;

___ b = *ptr;
```

dereference operator (*)

= value of the referenced variable

[✓ A. int]

B. int*

C. Compilation error

QUIZ

What will this code display?

```
int a = 45;
int* b = a;
```

A. 45

B. 0x10000

C. Compilation error

ANSWER

What will this code display?

```
int a = 45;
int* b = a;
```

tries to assign an integer to a pointer.

A. 45

B. 0x10000

[✓ C.] Compilation error

QUIZ

A pointer is a variable that contains as its value the ___________ of another variable.

A. size
B. address
C. value
D. type

ANSWER

A pointer is a variable that contains as its value the __________ of another variable.

A. size

[✓ B. address]

C. value

D. type

EXPLAIN

Wrap Up

Variable **definition**	`int num;`
Variable **initialization**	`num = 30;`
Address of a variable	`&num`
Pointer definition	`int* pointer;`
Pointer **initialization**	`pointer = &num;`
Print a pointer	`printf("%p", pointer);`
Access to the value of a pointer	`*pointer`

ACTIVITY

Let's **Try** !

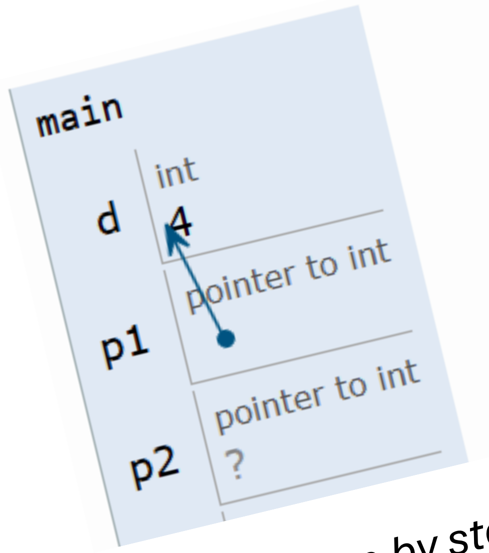
✓ Open the online debugger to run your code step by step and see the memory allocation.

YOUR MISSION

✓ Create a variable of type double *(ex: score)*
✓ **Print the address** of this variable

✓ Create a **pointer** to this variable
✓ **Print the address** of the pointer

✓ Change the **variable value** using the pointer
✓ **Print the value** of the variable



Observe step by step how memory is managed

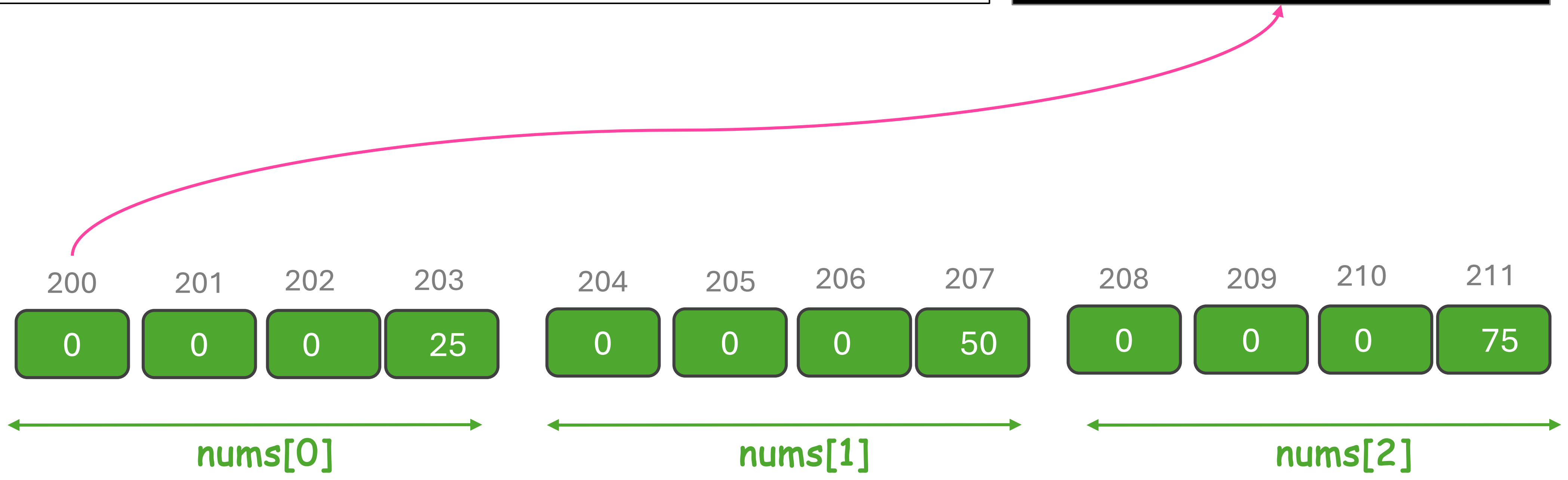
EXPLAIN

Array and pointers

An array name is nothing more than a pointer to the start of the array !

```
int nums[3] = {25, 50, 75};
printf("%p", nums)
```

```
0x200
```



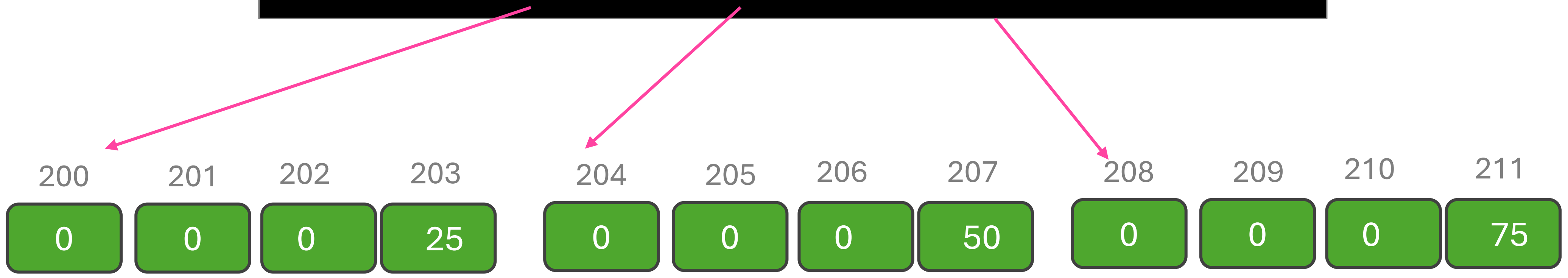
EXPLAIN

Array elements **address**

Let's print the **memory address** of each element

```
int myNumbers[3] = {25, 50, 75};
for (int i = 0; i < 3; i++) {
   printf("%p ", &myNumbers[i]);
}
```

```
0x200 0x204  0x208
```



What is the difference (in bytes) between 2 consecutive addresses of this array elements ? *Why* ?

EXPLAIN

Array element access using pointers

To access the array elements, we also can **increment the pointer**

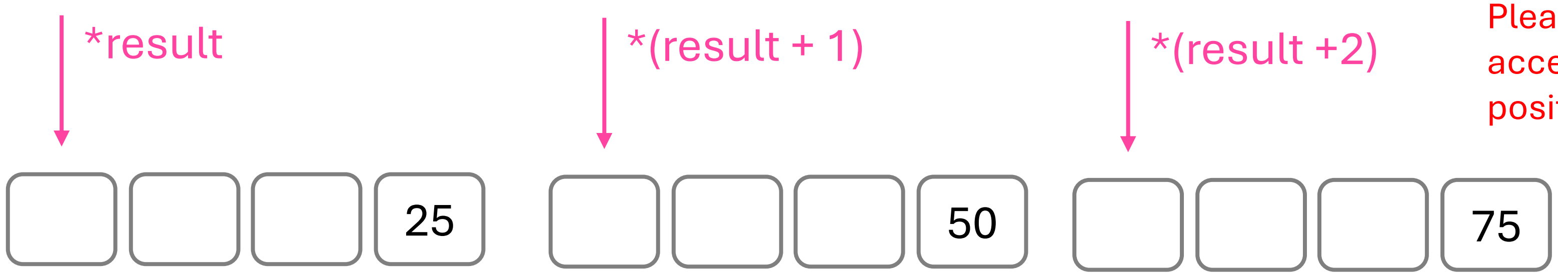
ACCESS USING []

```
int a = results[0];    // 25

int a = results[2];    // 75
```

ACCESS USING POINTERS

```
int a = *results;         // 25

int a = *(results + 2); // 75
```

Please use () when you want to access the value in a specific position.



```
int results[3] = {25, 50, 75};
```

QUIZ

How would you access the third element in myNumbers using pointers?

```
int myNumbers[4] = {25, 50, 75, 100};
```

A. myNumbers[3]
B. *myNumbers + 3
C. *(myNumbers + 2)
D. myNumbers[2]

ANSWER

How would you access the third element in myNumbers **using pointers**?

```
int myNumbers[4] = {25, 50, 75, 100};
```

A. myNumbers[3]

B. *myNumbers + 3

[✓ C.] *(myNumbers + 2)

D. myNumbers[2]

ACTIVITY

Let's **Try** !

✓ Open the online debugger to run your code step by step and see the memory allocation

YOUR MISSION

✓ Display **all array values** using **pointer access**

✓ Change the fourth element of the array **using pointers**

```
Before change:
10
11
12
13
After change
10
11
12
99
```

REFLECT

Functions and pointers

What will each code display?

CASE 1

```
void add5(int value) {
   value += 5;
}

int main() {
  int a = 45;
  add5(a);


  printf("%d", a);
  return 0;
}
```

CASE 2

```
void add5(int* value) {
   *value += 5;
}

int main() {
  int a = 45;
  add5(&a);


  printf("%d", a);
  return 0;
}
```

ANSWER

Functions and pointers

What will each code display?

CASE 1

```
void add5(int value) {
   value += 5;
}

int main() {
  int a = 45;
  add5(a);


  printf("%d", a);
  return 0;
}
```

Print 45 as the variable is passed by **value** (COPY)

CASE 2

```
void add5(int* value) {
   *value += 5;
}

int main() {
  int a = 45;
  add5(&a);


  printf("%d", a);
  return 0;
}
```

Print 50 as the variable is passed by **reference**

EXPLAIN

Pass by **reference** or by **value**

pass by reference
cup =
fillCup( )

pass by value
cup =
fillCup( )

LESS MEMORY USAGE

When passing arguments by reference, the function has the capability to **modify the value**

When passing arguments by value, C **makes a copy** of the argument for the receiving function to use

EXPLAIN

Pass by **const reference**

Using a const with a pointer allows **read-only access** while still passing by reference

MORE SECURE

```
void add5(int* value) {
    *value += 5;
}

int main() {
  int a = 45;
  add5(&a);

  printf("%d", a);
  return 0;
}
```

```
void add5(const int* value) {
    *value += 5;
}

int main() {
  int a = 45;
  add5(&a);

  printf("%d", a);
  return 0;
}
```

❌ We cannot change the value it points to

EXPLAIN

Wrap Up

- ✓ **Pointers** store **memory addresses** of other variables.
- ✓ Use ***** to declare a **pointer**, and **&** to get a variable's **address**.
- ✓ Initialize **pointers** with a variable's **address** or **NULL**.
- ✓ Use **%p** to print pointer addresses.
- ✓ Pointers enable pass-by-reference.
- ✓ Assigning an array to a pointer stores the **address** of its first element.
- ✓ **const** with pointers allows read-only access while still passing by reference.

What you **know now**

- ✓ Understand **memory addresses** and **pointers**.
- ✓ Declare and initialize **pointers**.
- ✓ Use the **address (&)** and **indirection (*)** pointer operators.
- ✓ Use **pointer arithmetic** to iterate through arrays.
- ✓ Use pointers to pass arguments to functions **by reference**.

Go further after the class…

To understand the basics:
https://www.w3schools.com/c/c_memory_address.php

https://www.w3schools.com/c/c_pointers.php

http://w3schools.com/c/c_pointers_arrays.php

To learn a bit more about pointer and references:
https://www.shiksha.com/online-courses/articles/difference-between-pointer-and-reference-blogId-155435