Computer System Organization Recitation [Spring 2019]

CSCI-UA 201-002

R4: C types, pointer

Some slides based on Chien-Chin Huang's Spring 2018 CSO recitation

Request grace days for labs

- Lab1 is due in a week
- Detailed instructions to request grace day
  - https://piazza.com/class/jqwimku57oj7bj?cid=36
- You **MUST** request grace days within 5 days from lab1 due date
- Try to save grace days for future harder labs

Lab1 output format checker

- Make sure you strictly follow output format requirements
- Use output_format_checker to check your output format
  - https://piazza.com/class/jqwimku57oj7bj?cid=37
- If you have any questions, come to office hour to speak to instructor

Today’s topic

- Basic C
  - types in C
  - pointer
- Today’s exercise

Tip to learn C

- Think about everything from memory perspective
- Everything is stored in memory:
  - data (variable, arrays, …)
  - program (functions)
- Think about how each line of your code is reflected in memory

Types in C

type	size (bytes)	example
(unsigned) char	1	char c = 12 char c = ‘a’
(unsigned) short	2	short s = 12
(unsigned) int	4	int i = 1
(unsigned) long	8	long l = 1
float	4	float f = 1.0
double	8	double d = 1.0
pointer	8	int *x = &i

C is a strongly-typed language

- You MUST declare type for each variable
  - a=10; What's the default type? **Compilation Error**

Why data type is needed?

- Two pieces of necessary information for a type
  - storage: size of data, domain (i.e. possible values a data type can take)
  - computation: possible operations on this data type

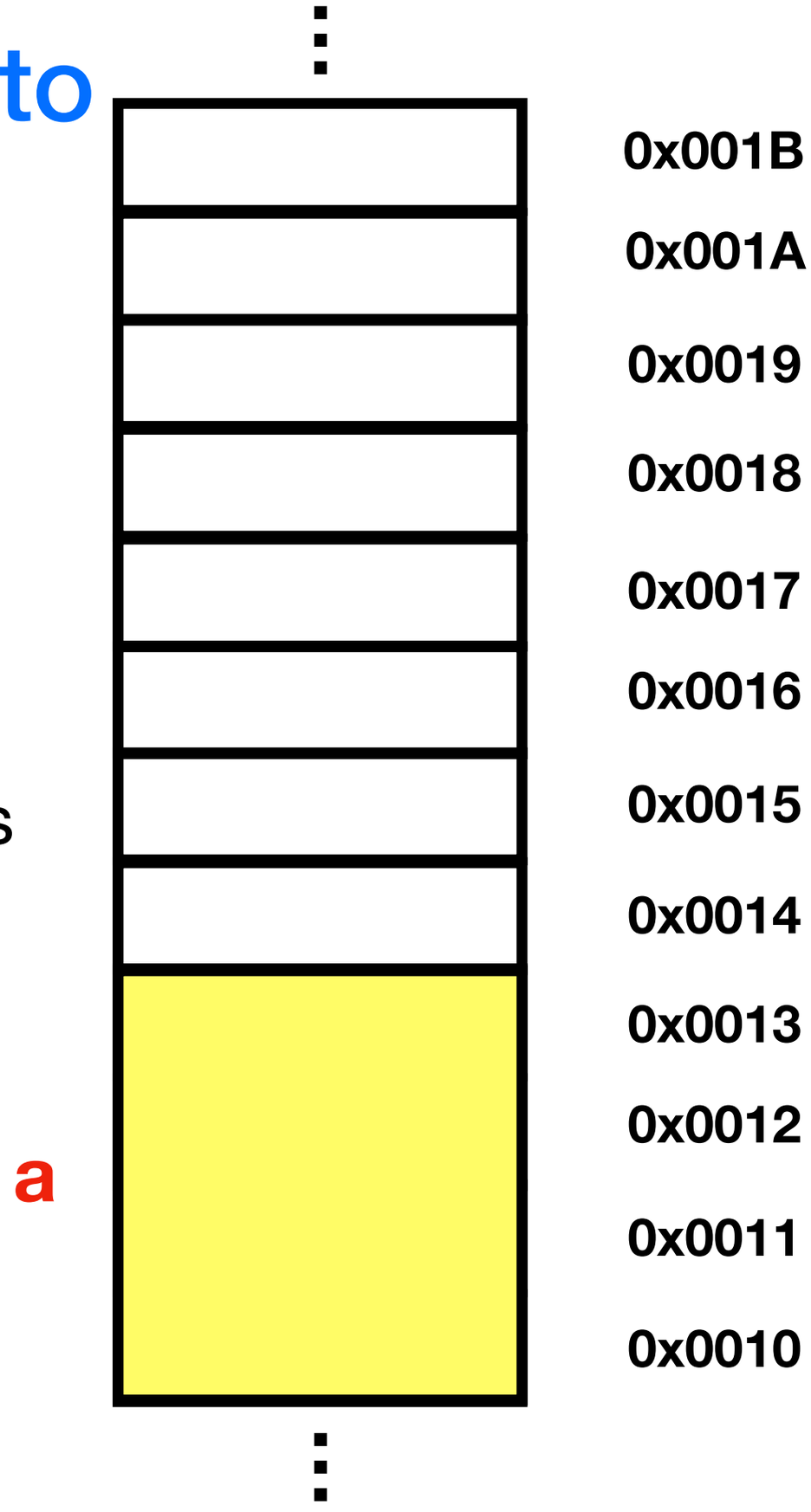
```
int a = 0;

// integer type
// 4 bytes, value range -2^31 ~ 2^31-1
// operations: + - * / %, etc
```

How does that connect to memory

int a = 10;

- When creating variable a,
  - 4 bytes space will be allocated
  - compiler memorizes the address and type of the variable

Name	Starting Address	Type
a	0x0010	int

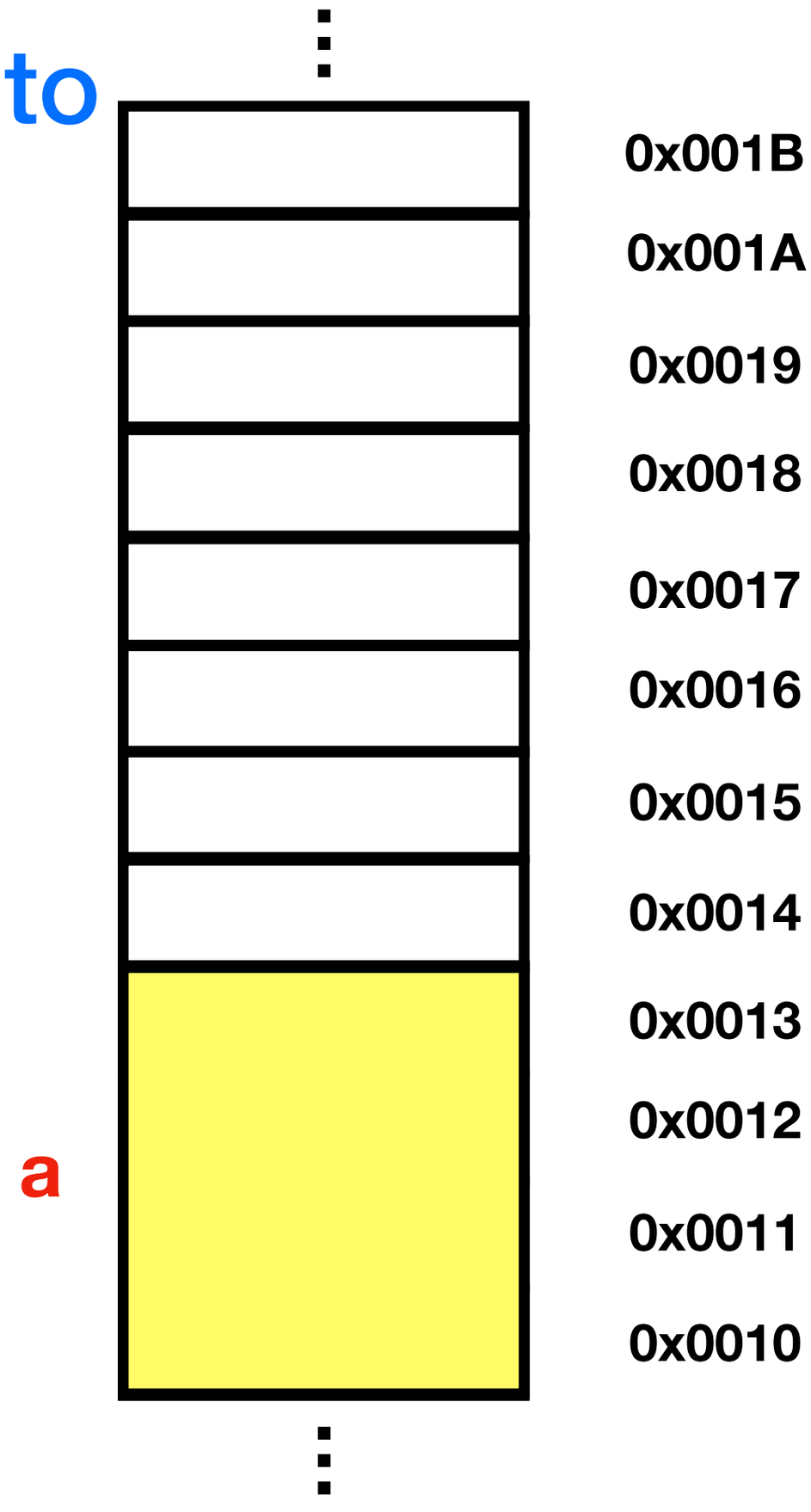


How does that connect to memory

Name	Starting Address	Type
a	0x0010	int

a = a + 1;

- When a variable is referenced:
  - table lookup to get address and type (address 0x0010, type int)
  - load the value from the memory address (value 10)
  - store value back to the memory address (write 11)



How to handle operations that involve different types?

- What's the result of the following code snippet?

```
int a = 10;
int b = 1.8;
int c = a + b;
printf("%d\n", c);
```

- Many operations can only handle operands of the same data type
  - Type conversion

Type conversion

- Implicit conversions (pre-defined rules in compiler)
  - integer promotion **unsigned a = 10; a > -10**
  - integer -> float -> double
  - google and read specifications

- Explicit conversions
  - If no implicit rules available, or you want to control compiler's behavior, use explicit conversion

unsigned a = 10; (int)a > -10

How about assignment op?

- What actually happened on assignment (=)?

  int a = 10; int b = a;

- Memory copy
  - if the left operand and right operand have the same date type

- What if two operands have different types? Can we do memory copy?
  - They might have different size **int a = 10; short b = a;**
  - They might mean different things **int a[10]; int b = a;**

Type conversion for assignment op

- Implicit rules
  - between integer types: memory copy, truncate memory or pad 0 if needed **int a = 10; short b = a;**
  - from float to integer: fractional part is truncated
    - If the target type cannot represent the integer part of the float number, then that's **undefined behavior**

- Use explicit conversion to control compiler behavior

Pointer in C

- What is pointer?
  - A variable, of which the value is an address

Pointer in C

- What is pointer?
  - A variable, of which the value is an address
- Size of a pointer variable
  - 8 bytes (64 bits) on 64-bit machine

How to use pointer

- initialize a variable

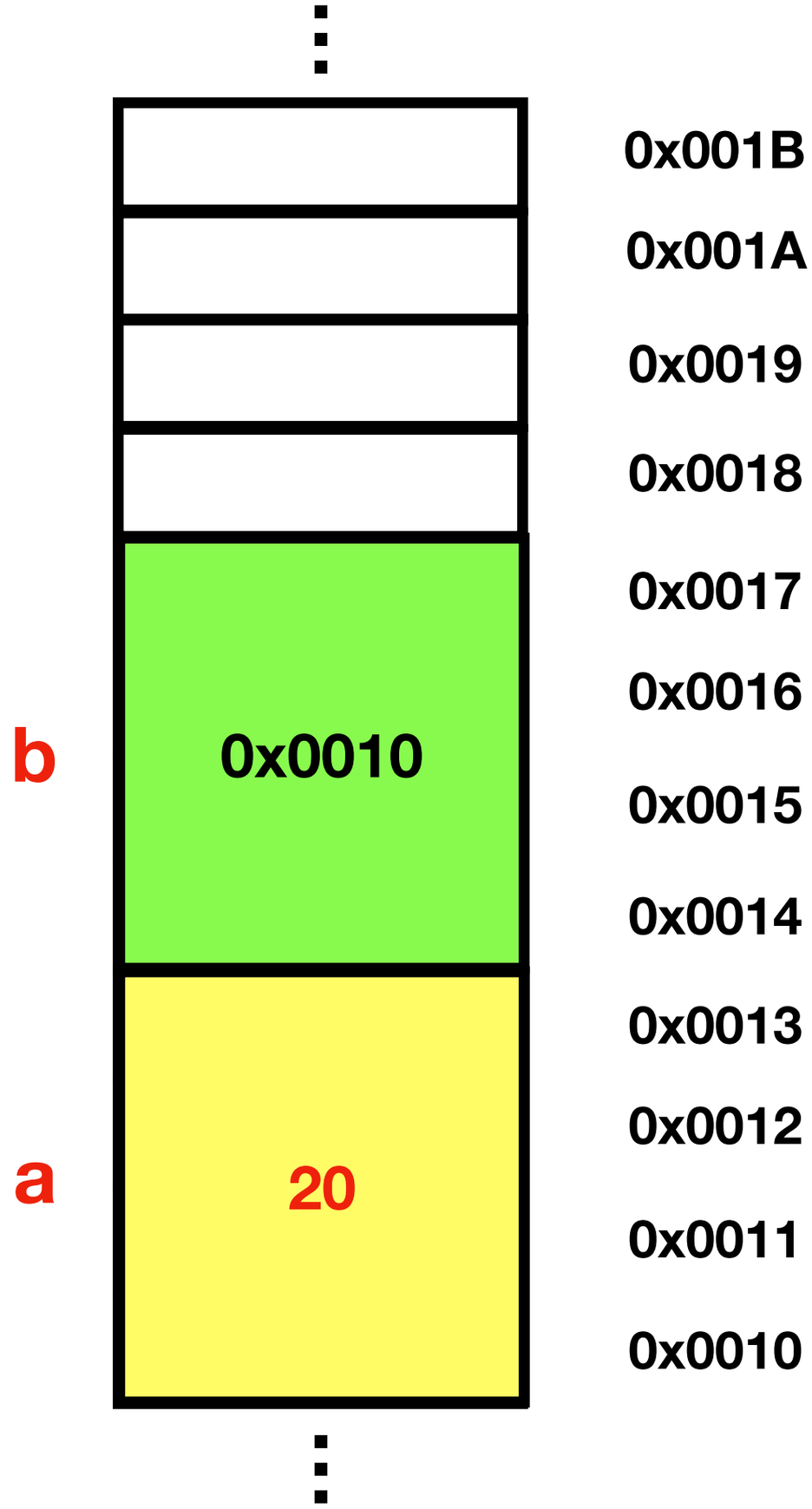
  int a = 10;

- initialize a pointer variable b to be the address of a

  int* b = &a;

- dereference b to change the value of a

  ***b = 20;**



Why do we need pointers?

- The example looks stupid
  - Why do we need to take so much trouble just to change the value of variable a?
- Isn’t python’s reference semantics sufficient?
  - especially for variable sharing purpose

Why do we need pointers?

- Read/write memory address are the most universal/ general operations
- Fine control on memory is needed for softwares like OS, drivers, compilers, and your video games …
- Your CPU can only use address to read and write memory
  - The concept of variable and reference only exists in high level programming language
  - When you compile your code into binary, everything is address

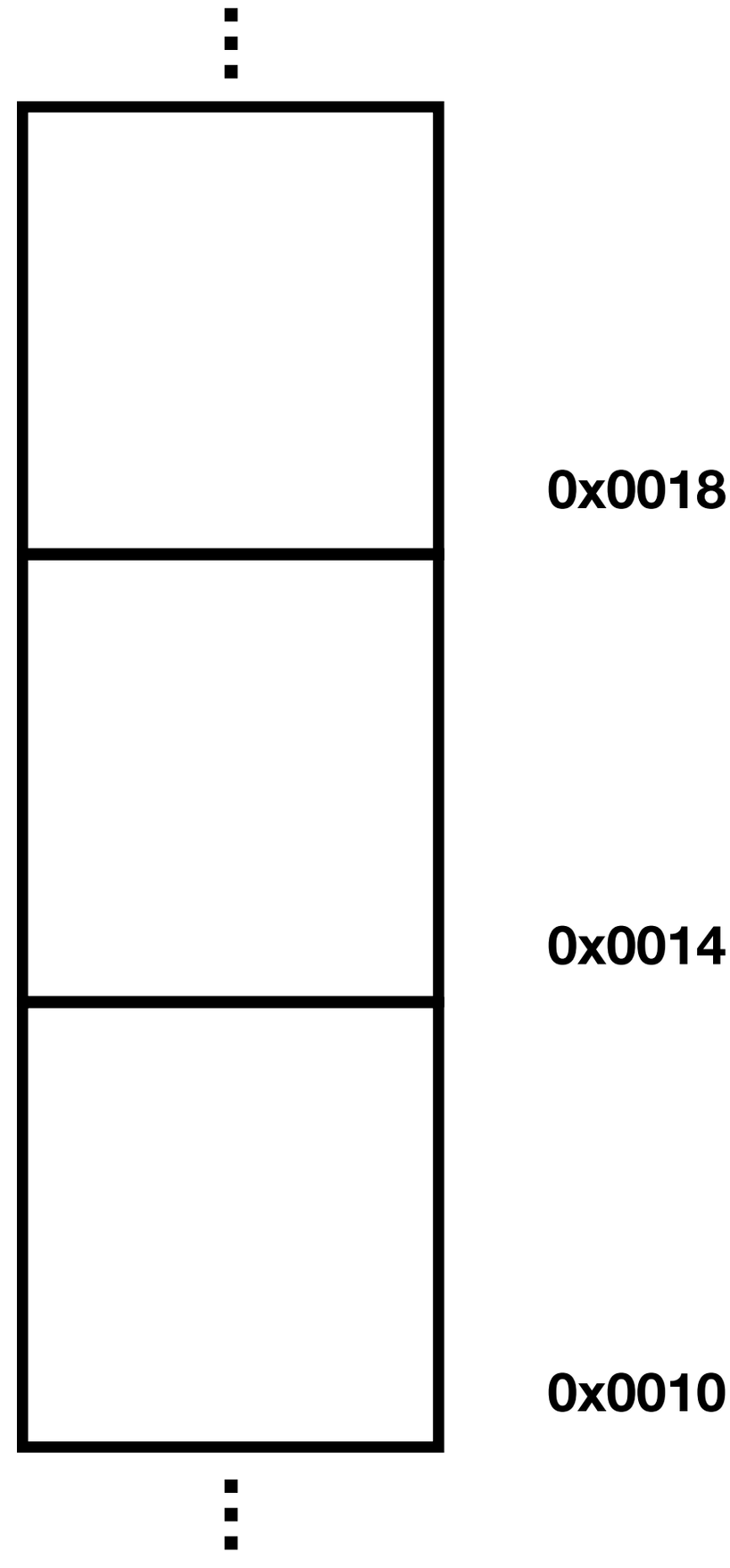
Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf(“%u”, num);
}
```

What’s the output of the program?

Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

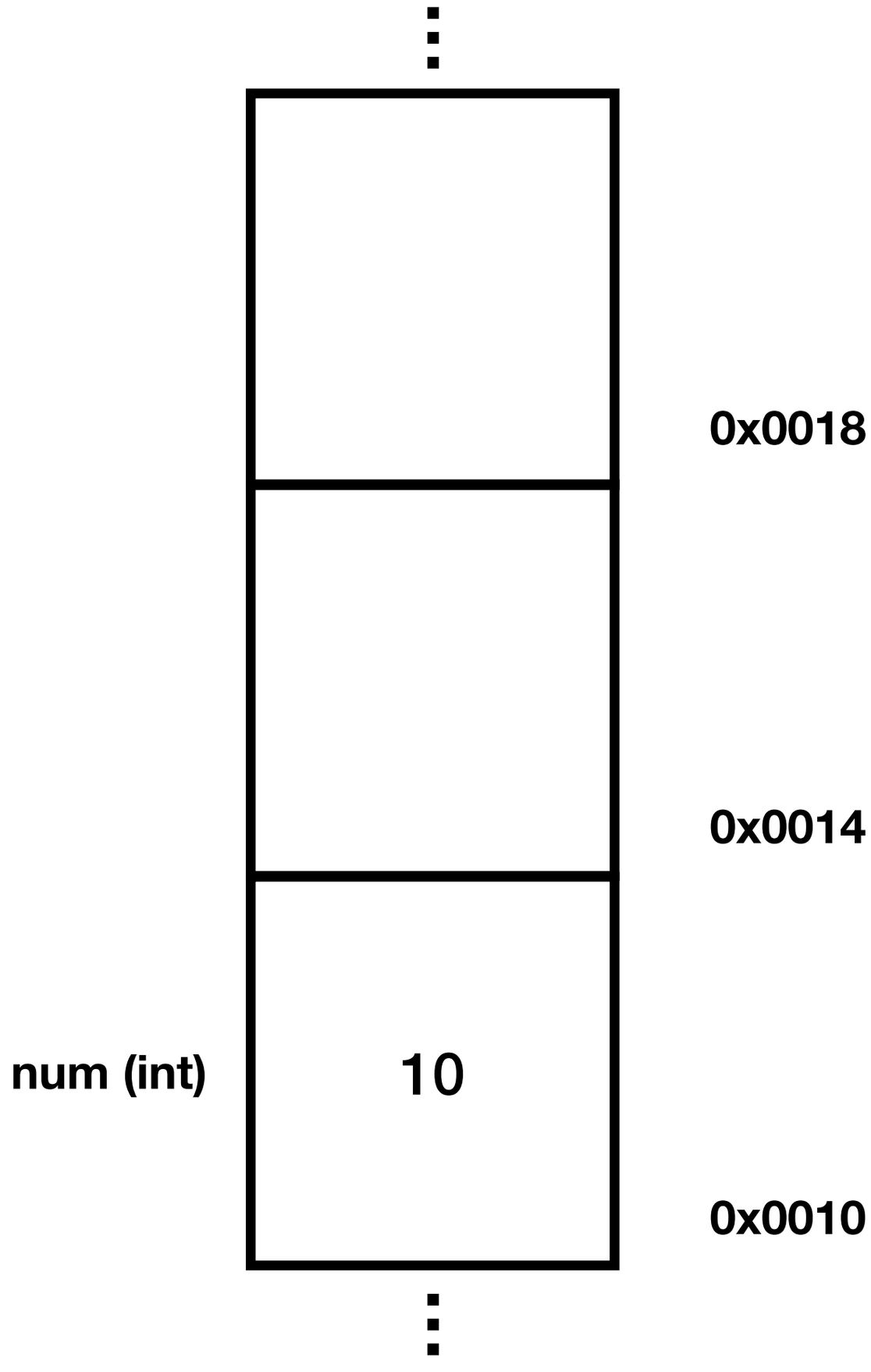


What's the output of the program?

Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

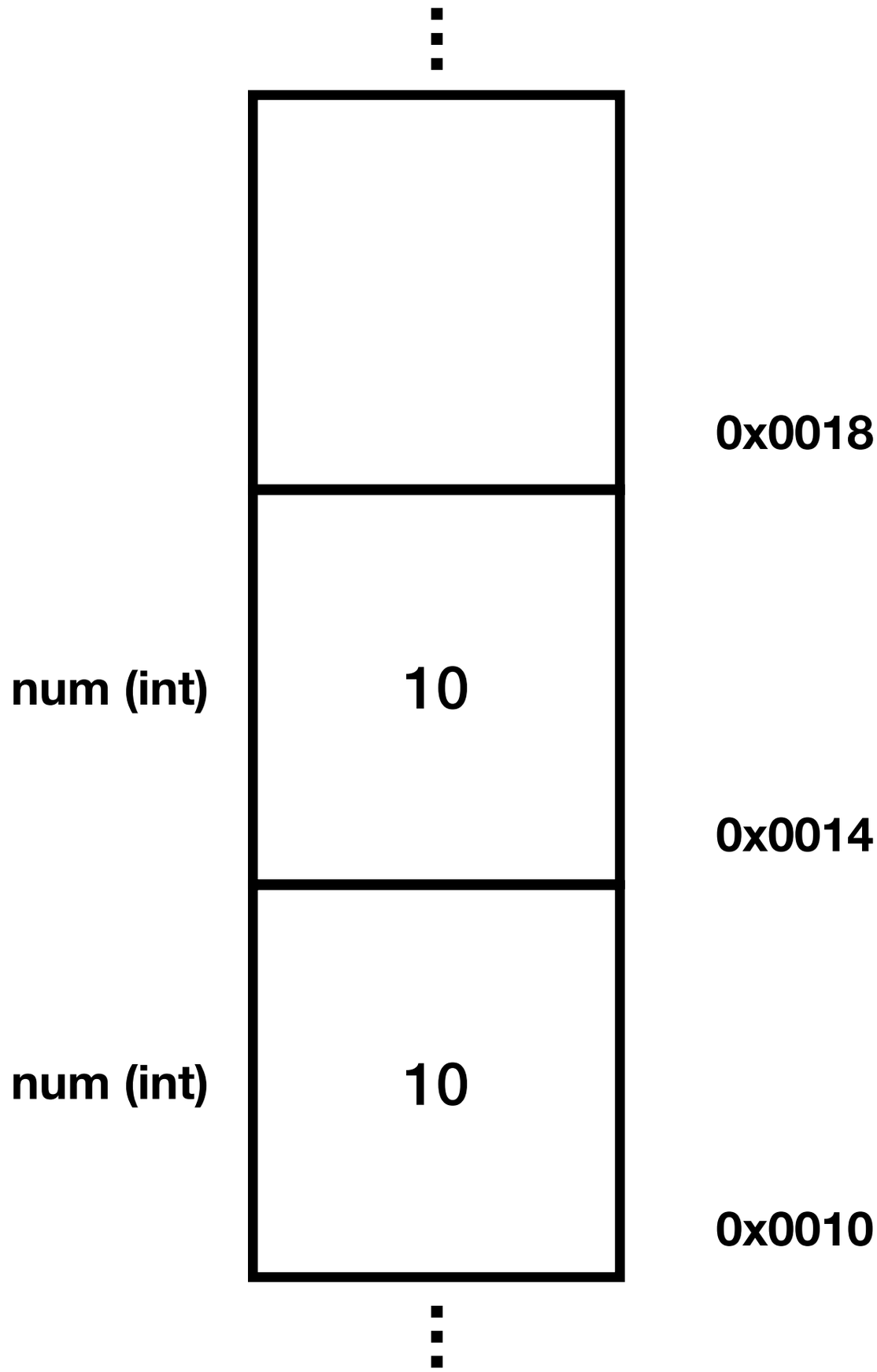
What's the output of the program?



Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

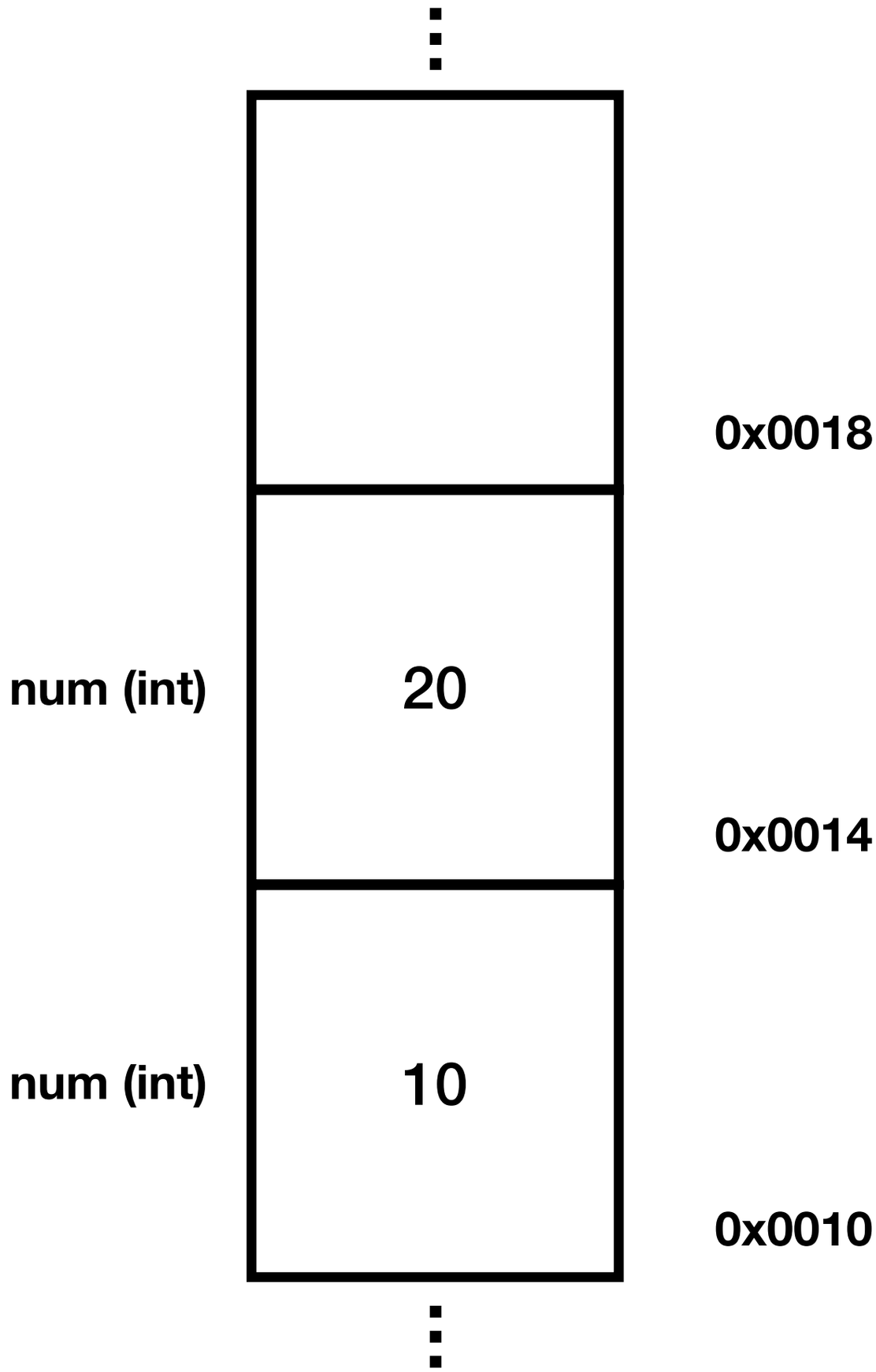
What's the output of the program?



Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

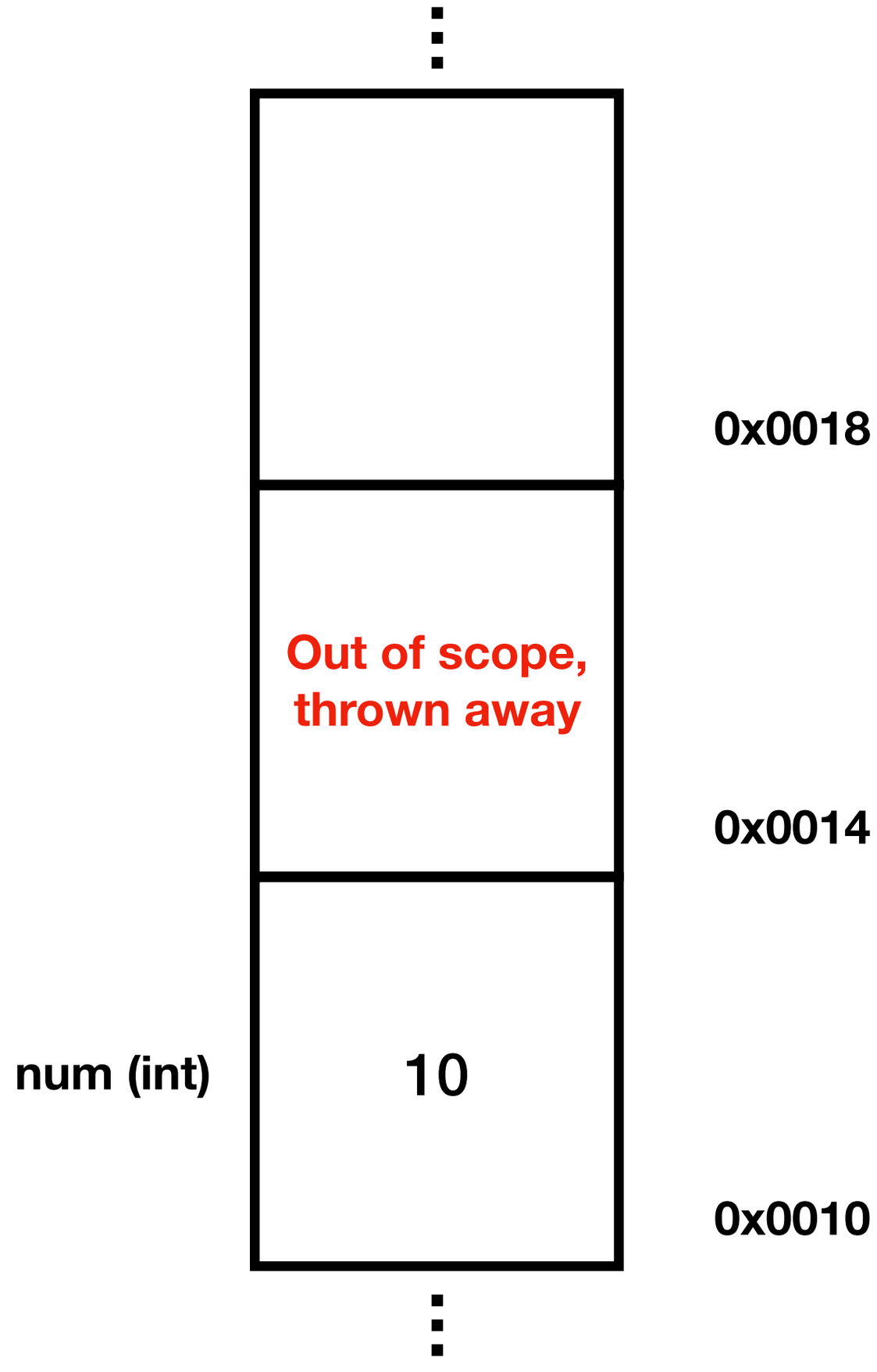
What's the output of the program?



Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

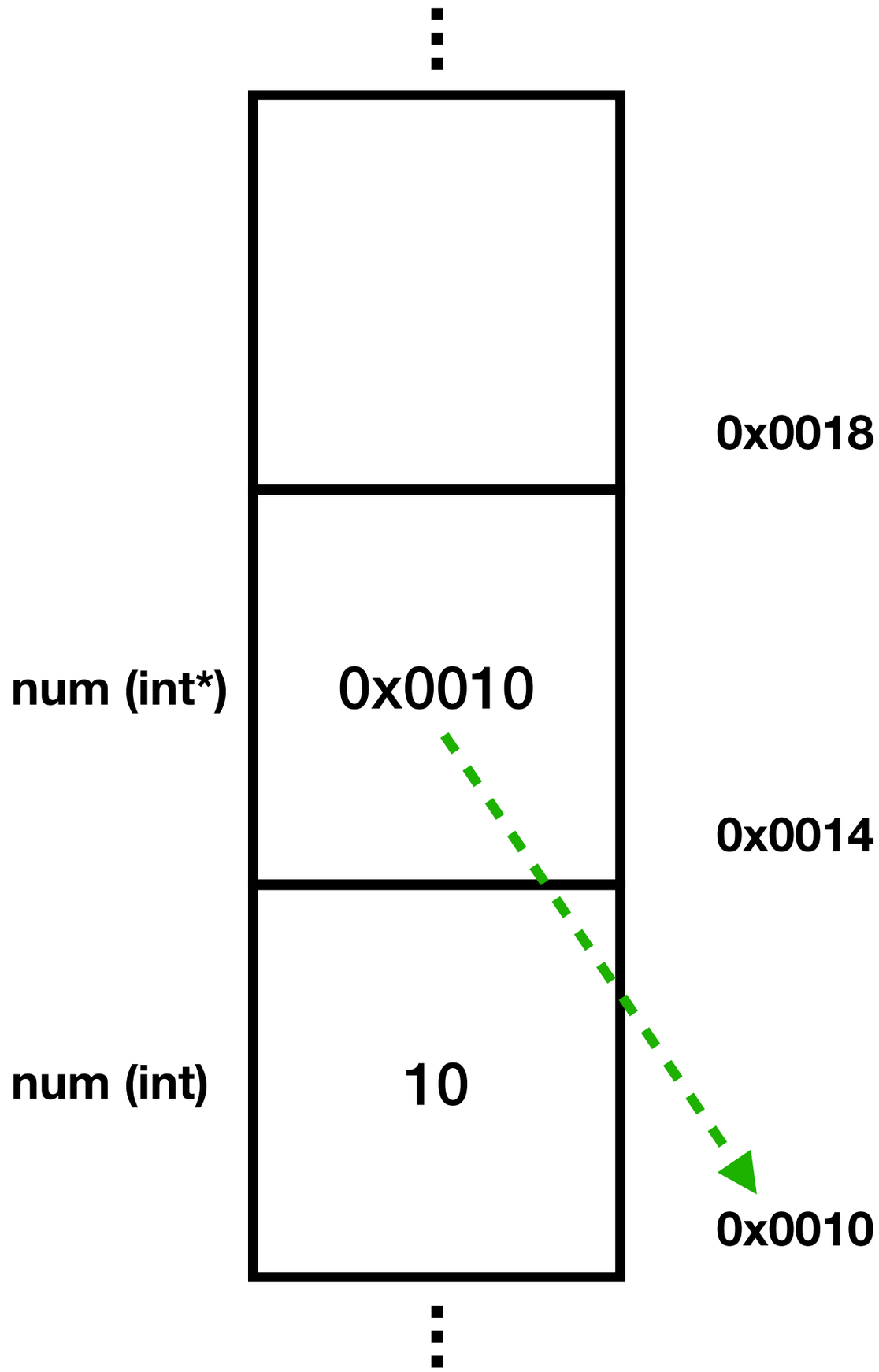
What's the output of the program?



Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

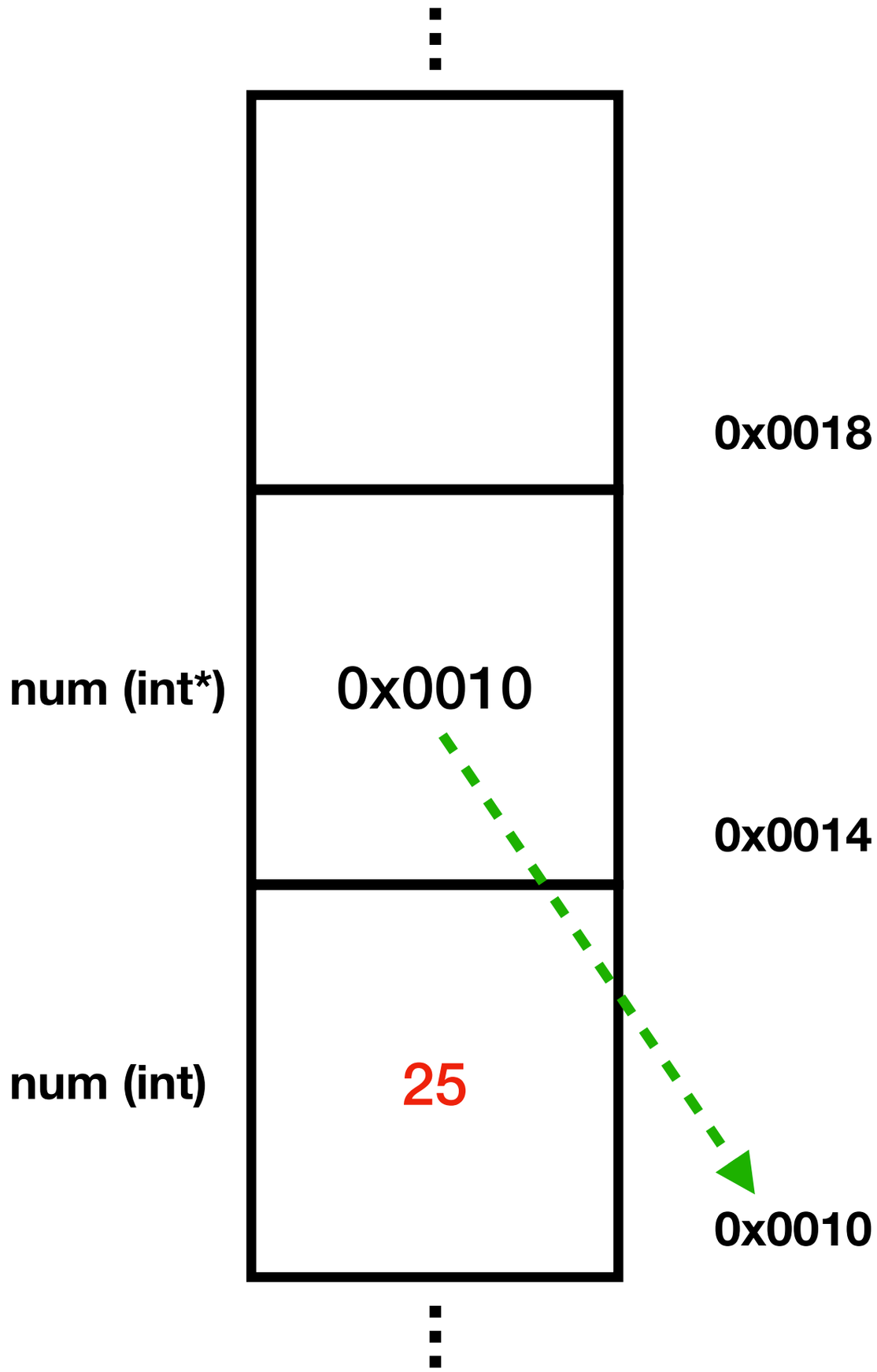
What's the output of the program?



Simple Exercise

```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

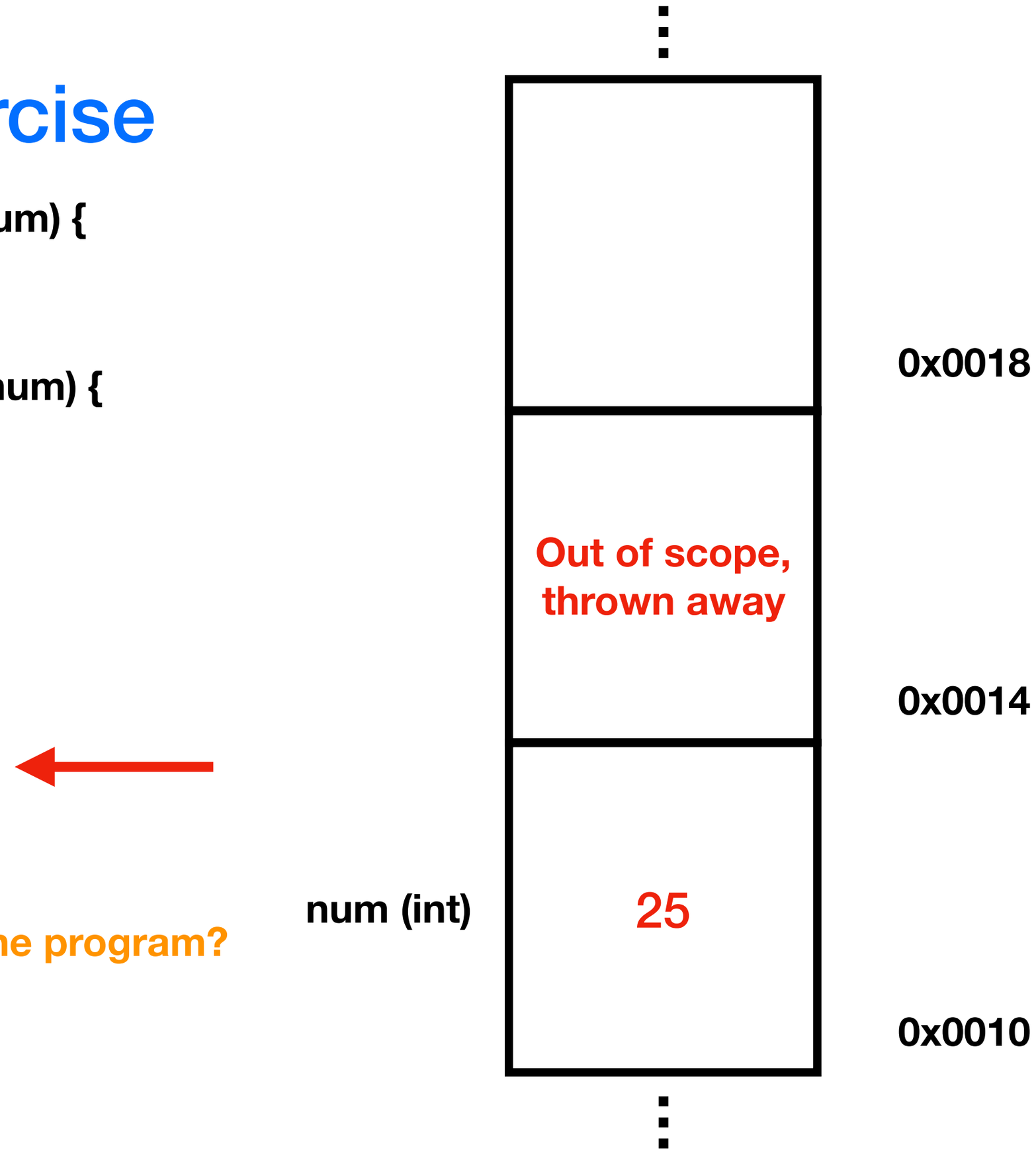
What's the output of the program?



Simple Exercise

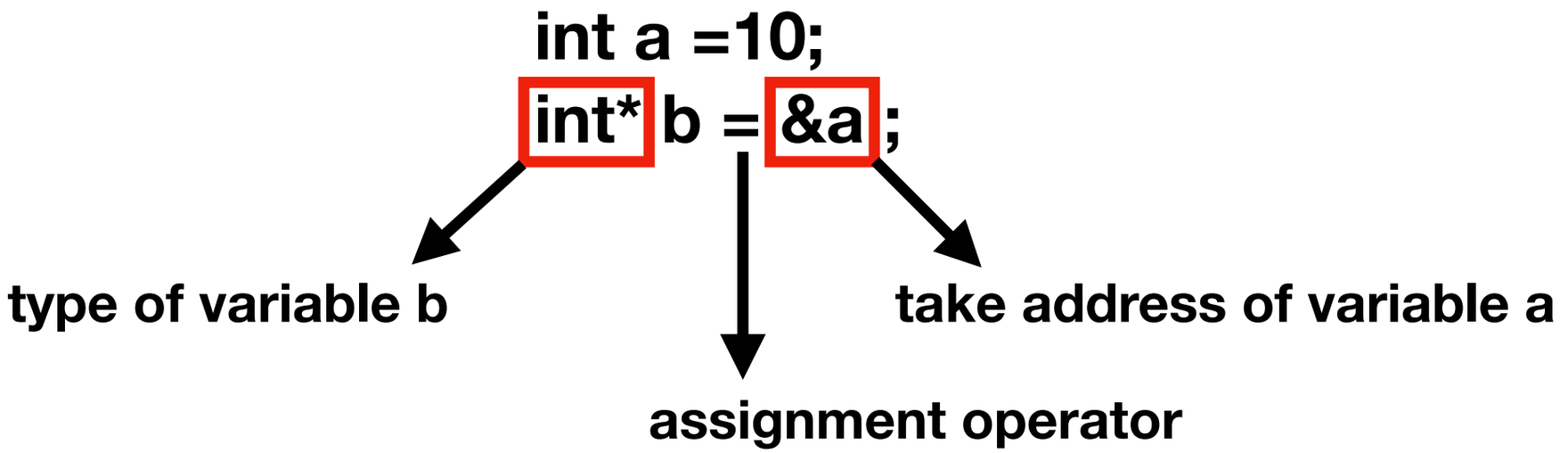
```
void func1(unsigned num) {
    num += 10;
}

void func2(unsigned* num) {
    *num += 15;
}

int main() {
    unsigned num = 10;
    func1(num);
    func2(&num);
    printf("%u", num);
}
```

What's the output of the program?



More about pointer type

- Define a pointer variable



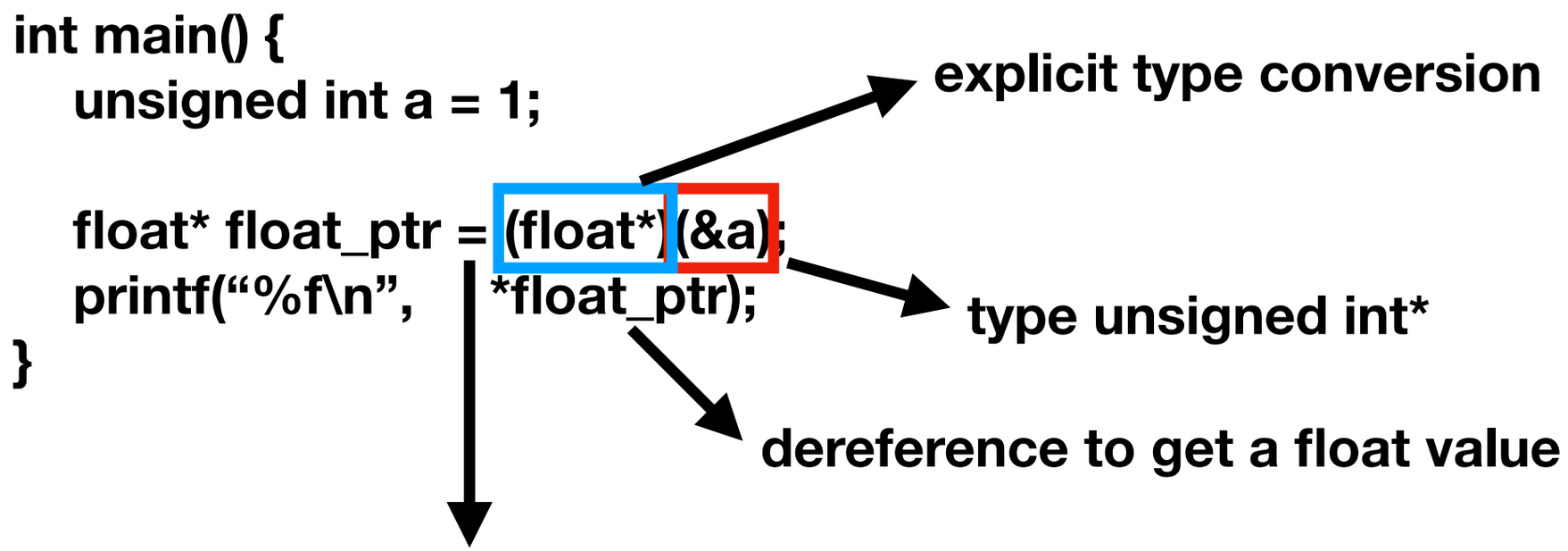
- Questions:
  - What's the type of expression &a?
  - How does compiler know the type of &a?

Pointer type

- Pointer variable stores an address (which is an unsigned long int)
- More questions to think about:
  - Since all address can be represented using unsigned long int, why do we need pointer type? (instead of directly using unsigned long int)
    - To define pointer operations like deference and pointer arithmetic
  - Why does pointer variable have different types? Like int*, char*, float*, etc
    - int* tells the compiler, the pointer variable is pointing to space storing an int

Type conversion for pointers

- pointer type conversion tells the compiler to interpret the memory space pointed to as a different type



Exercise

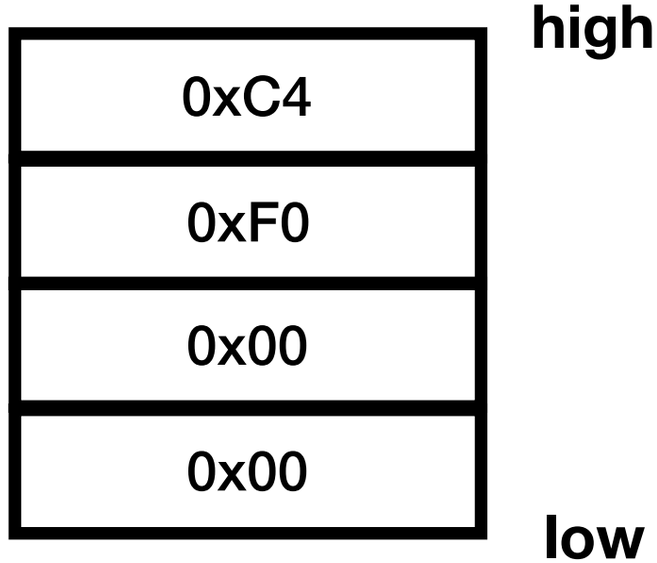
- If a float variable has the following memory layout, what's the decimal value?

Write a C program to get the answer

unsigned int a = 0xC4F00000;

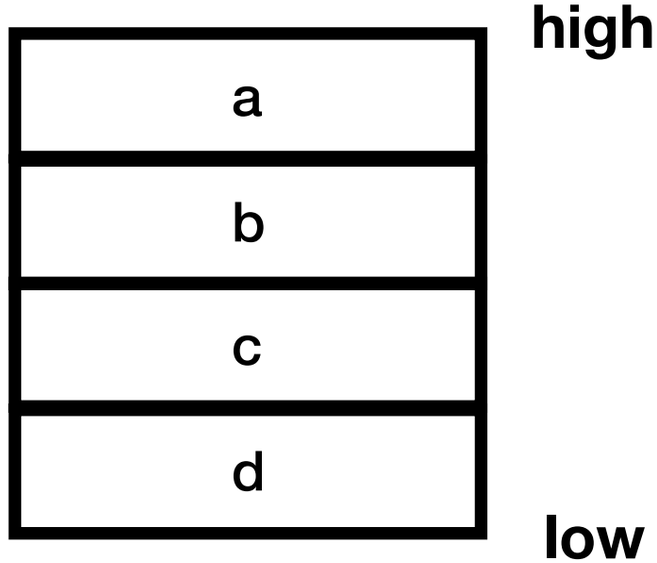
float* f_ptr = (float*)&a;

float f = *f_ptr;



Exercise

- If a float variable has decimal value 20.375, and the memory layout looks like the following, what's the hex representation of a, b, c, d



Write a C program to get the answer

```
float f = 20.375;

unsigned int* ptr = (unsigned int*)&f;

 printf("%.8X\n", *ptr);
```

void*

- What if I only know/need the memory address but not the type information?

- Pointer of type void* means pointer that points to a void type (no type needed)
  - Only address value is kept

- void* can be implicitly converted to any other pointer type, vice versa
  - NULL is of type void*
  - int* a = NULL;

Today’s exercises

- Two simple exercises