CTF of C Programming

Kosi Nwabueze <kosinw@mit.edu>

Spring 2026

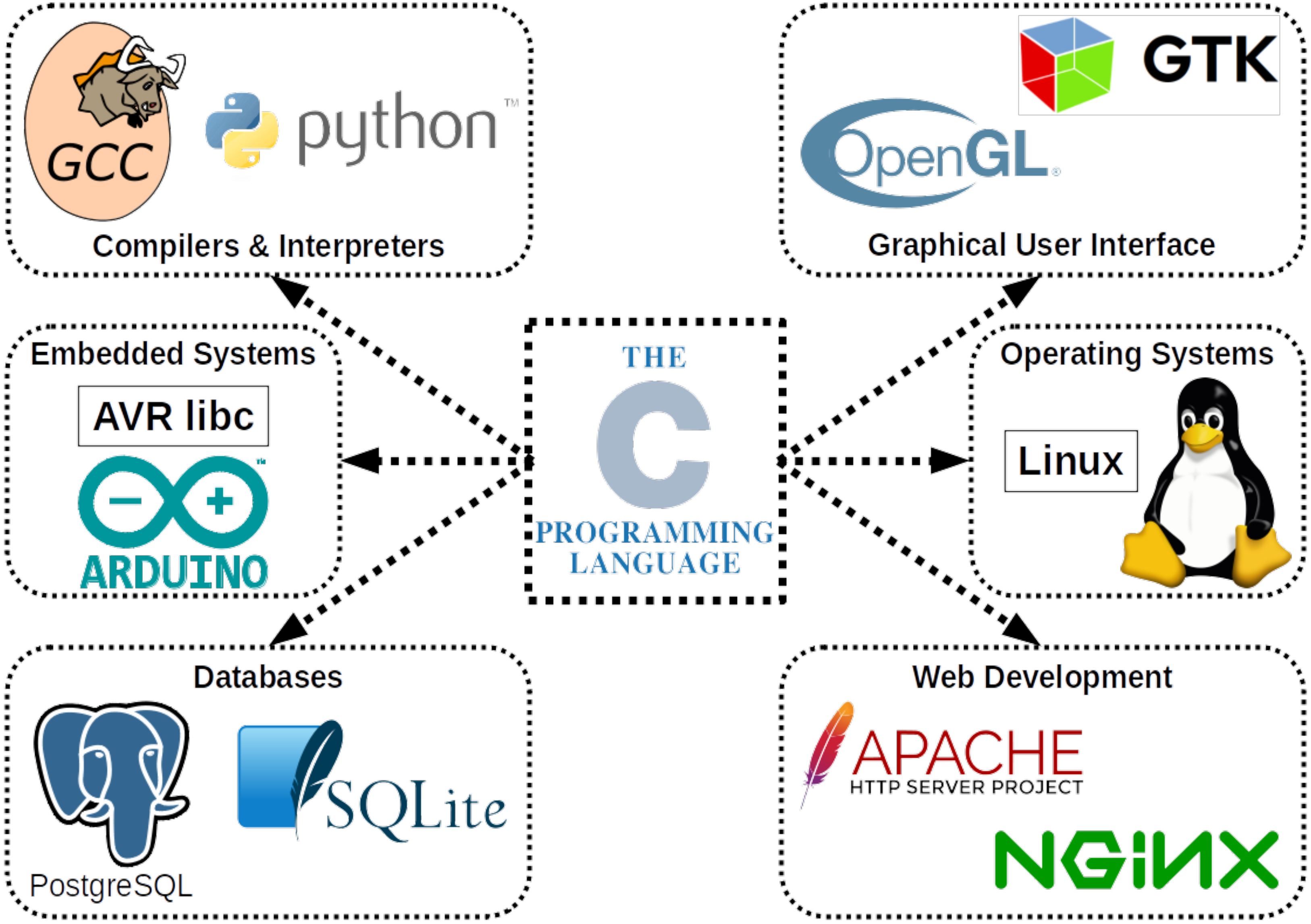


Image: https://commons.wikimedia.org/w/index.php?curid=109712425

Crash Course

```c
#include <stdio.h>
#define MAGIC_NUM 5

void sayHello(int helloNum) {
  printf("Hello World! The addition sum is: %d\n", helloNum);
}

int main(void) {
  int result = 1 + MAGIC_NUM;
  sayHello(result);
  return 0;
}
```

Crash Course

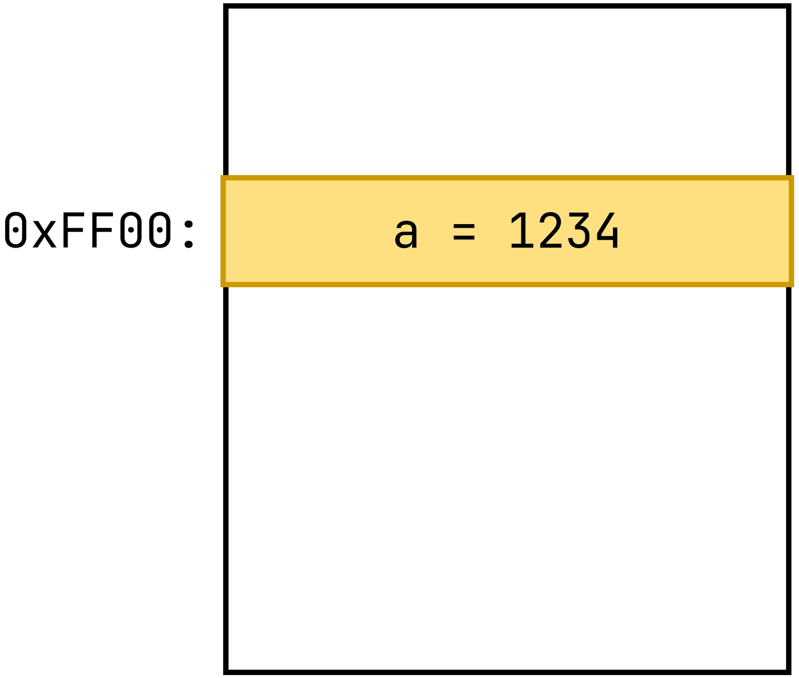
```c
#include <stdio.h>
#define MAGIC_NUM 5

void sayHello(int helloNum) {
  printf("Hello World! The addition sum is: %d\n", helloNum);
}

int main(void) {
  int result = 1 + MAGIC_NUM;
  sayHello(result);
  return 0;
}
```

Crash Course

```c
#include <stdio.h>
#define MAGIC_NUM 5

void sayHello(int helloNum) {
  printf("Hello World! The addition sum is: %d\n", helloNum);
}

int main(void) {
  int result = 1 + MAGIC_NUM;
  sayHello(result);
  return 0;
}
```

Pointers

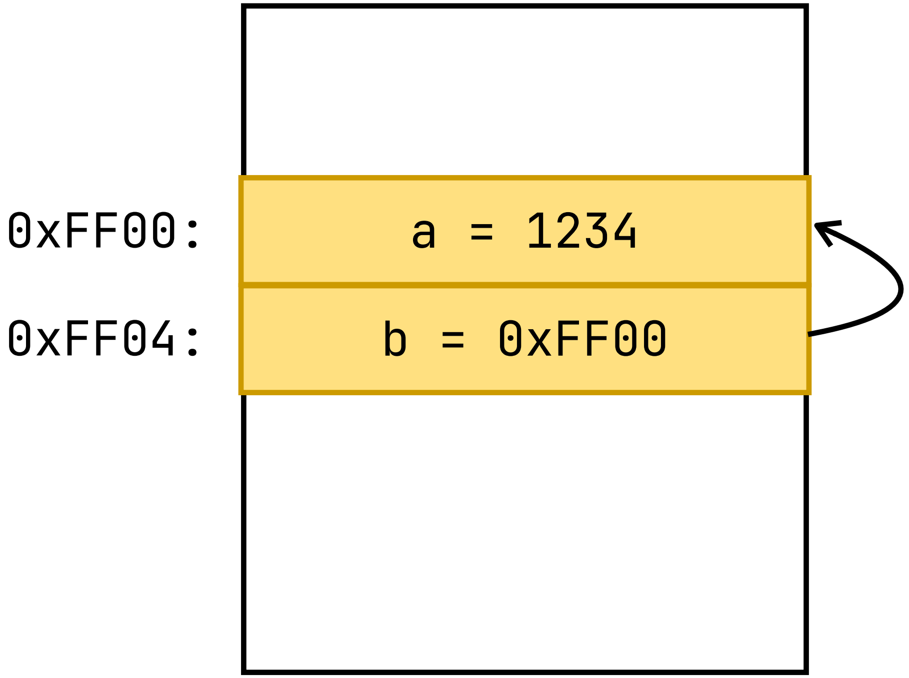
```
void example_method() {
   int a = 1234; // (1)
   int *b = &a;  // (2)
   *b = 9876;    // (3)
}
```



(1)

Pointers

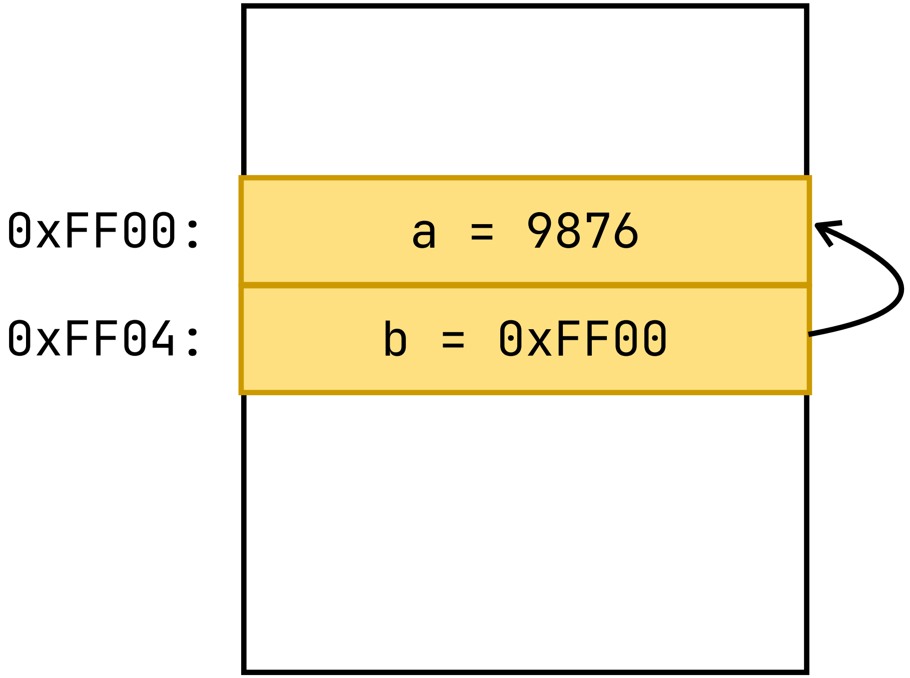
```
void example_method() {
  int a = 1234; // (1)
  int *b = &a;  // (2)
  *b = 9876;    // (3)
}
```



(2)

Pointers

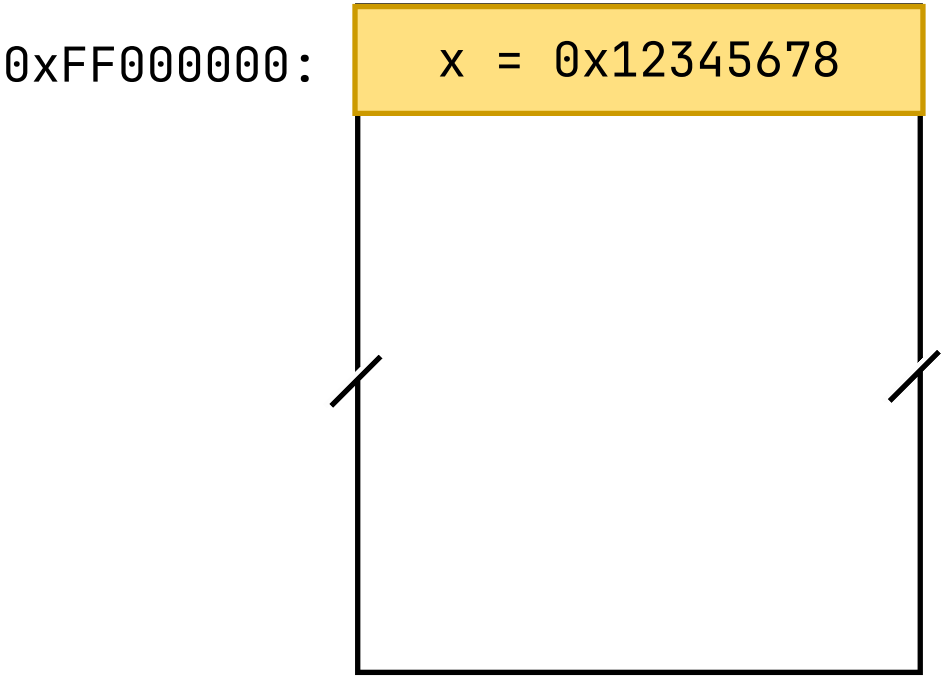
```
void example_method() {
   int a = 1234; // (1)
   int *b = &a;  // (2)
   *b = 9876;    // (3)
}
```



(3)

Casting

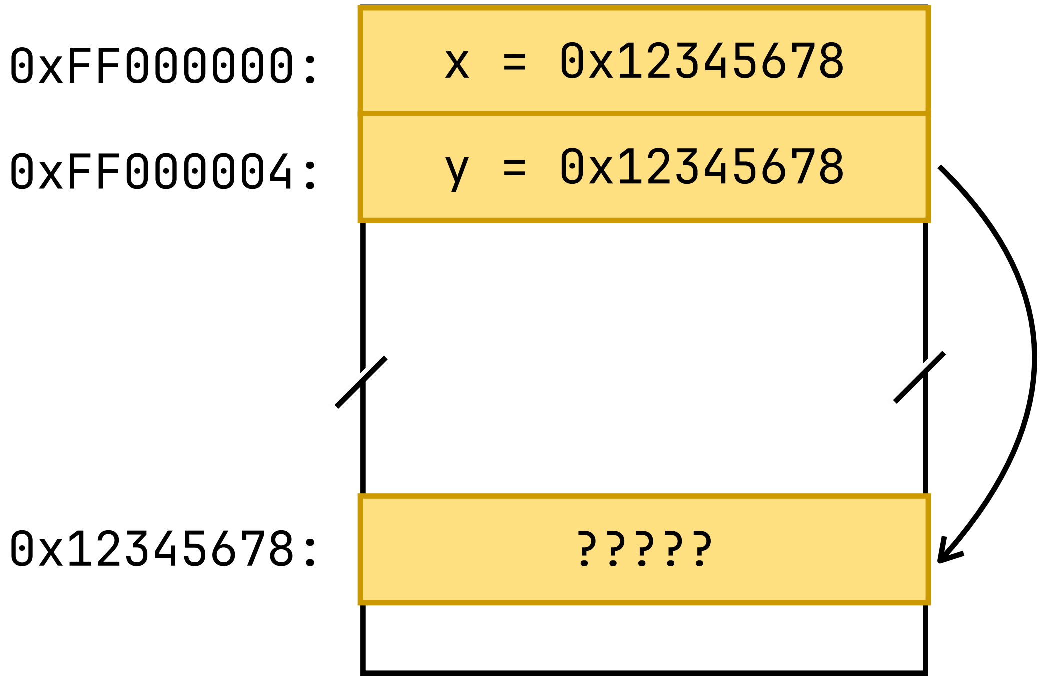
```
void example_method() {
    uint64_t x = 0x12345678;
    uint8_t *y = (uint8_t *) x;
    uint8_t z = *y;
}
```



(1)

Casting

```
void example_method() {
    uint64_t x = 0x12345678;
    uint8_t *y = (uint8_t *) x;
    uint8_t z = *y;
}
```



0xFF000000: x = 0x12345678

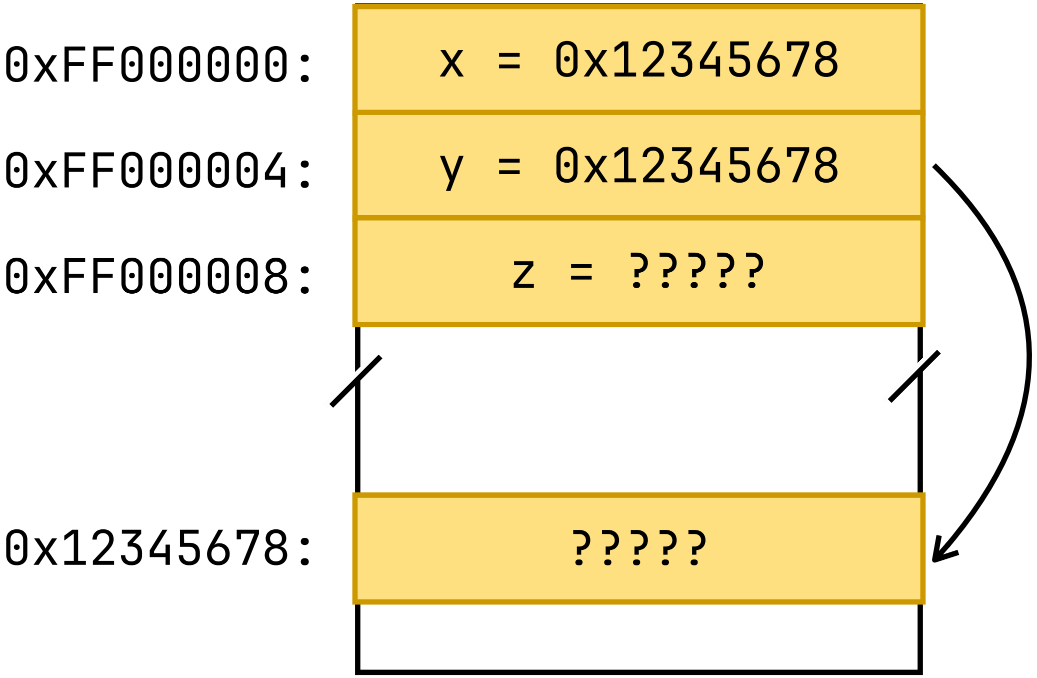
0xFF000004: y = 0x12345678

0x12345678: ?????

(2)

Casting

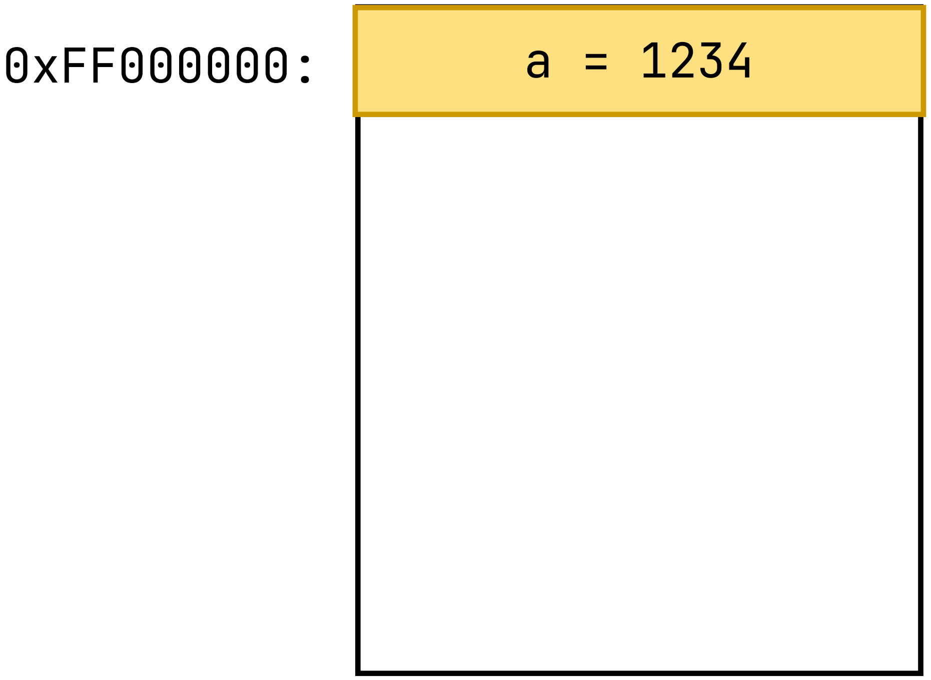
```
void example_method() {
    uint64_t x = 0x12345678;
    uint8_t *y = (uint8_t *) x;
    uint8_t z = *y;
}
```



(3)

Double Pointers

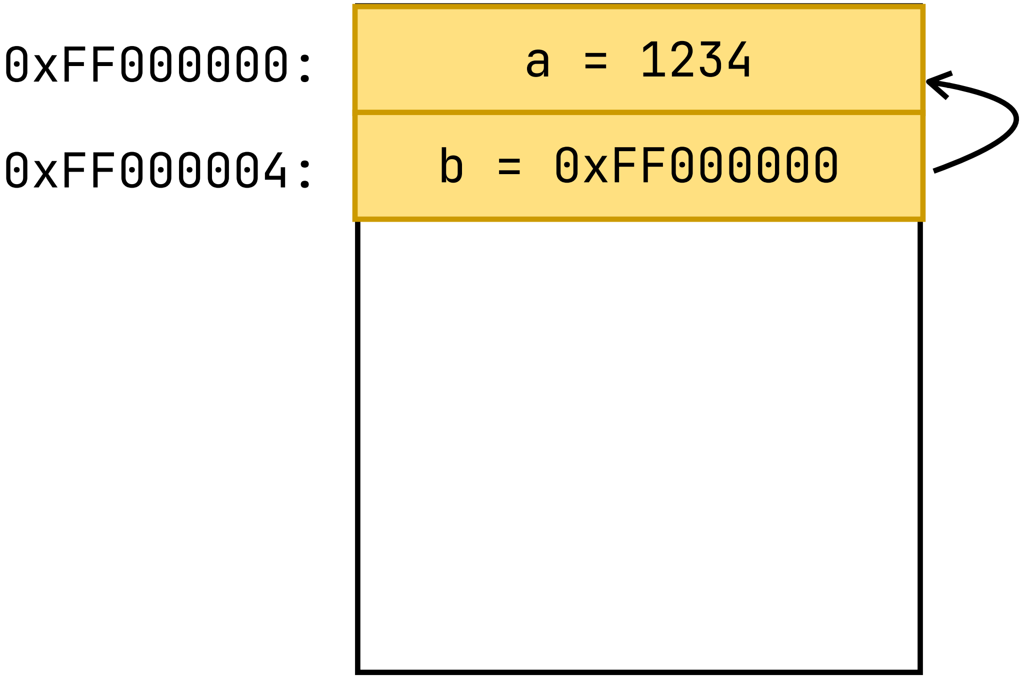
```c
int a = 1234;
int *b = &a;
int **c = &b;
```



(1)

Double Pointers

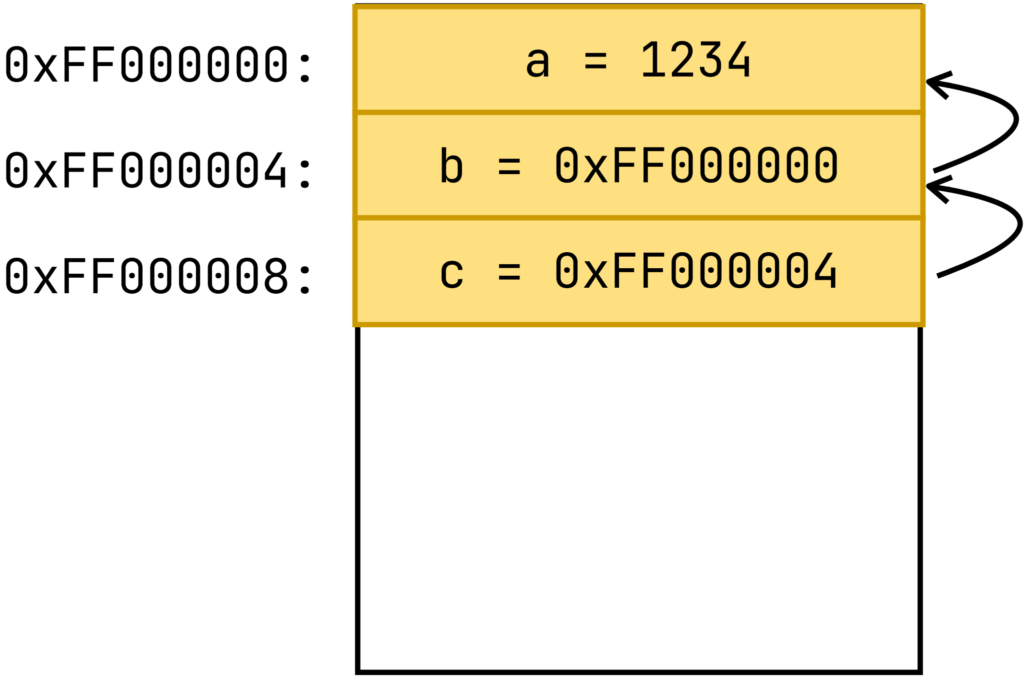
```
int a = 1234;
int *b = &a;
int **c = &b;
```



(2)

Double Pointers

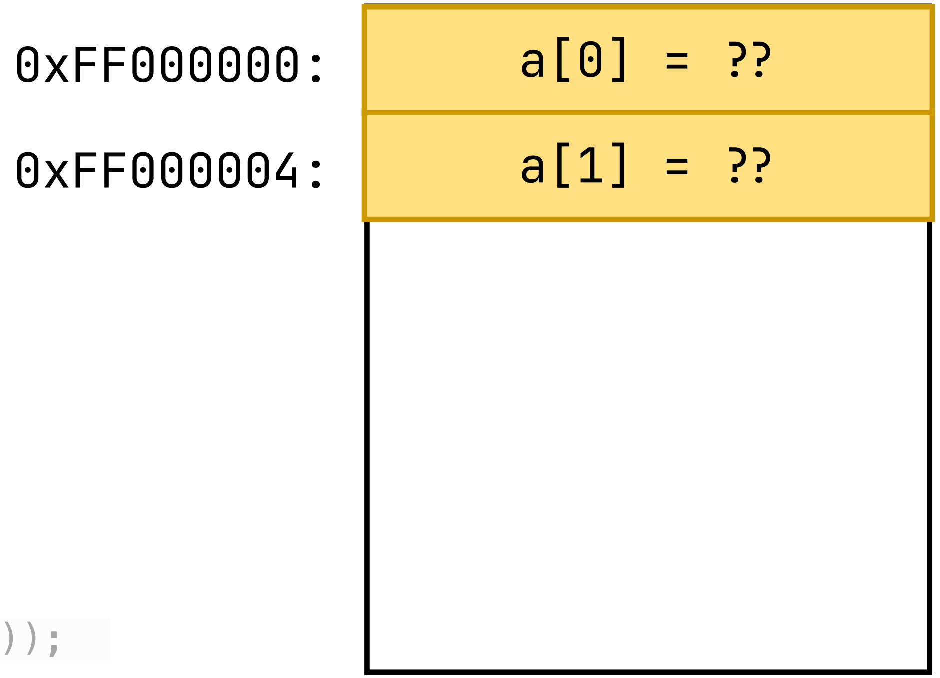
```
int a = 1234;
int *b = &a;
int **c = &b;
```



(3)

Arrays

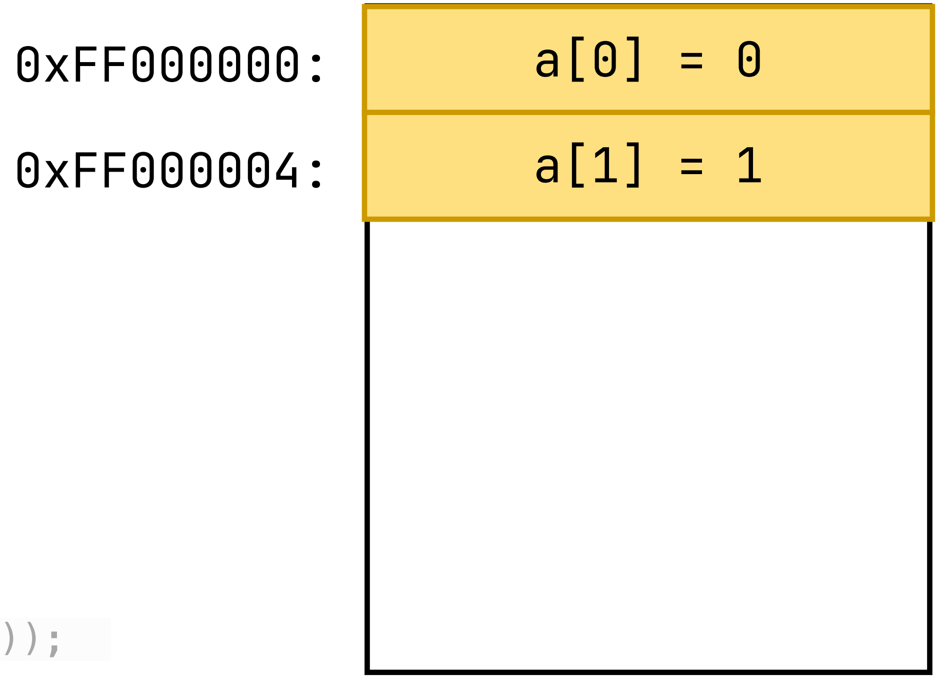
```
void example_method() {
  int a[2];

  a[0] = 0;
  a[1] = 1;

  printf("The first element of a is: %d\n", a[0]);
  printf("The second element of a is: %d\n", a[1]);

  printf("The first element of a is: %d\n", *(a));
  printf("The second element of a is: %d\n", *(a + 1));
}
```



(1)

Arrays

```
void example_method() {
  int a[2];

  a[0] = 0;
  a[1] = 1;

  printf("The first element of a is: %d\n", a[0]);
  printf("The second element of a is: %d\n", a[1]);

  printf("The first element of a is: %d\n", *(a));
  printf("The second element of a is: %d\n", *(a + 1));
}
```



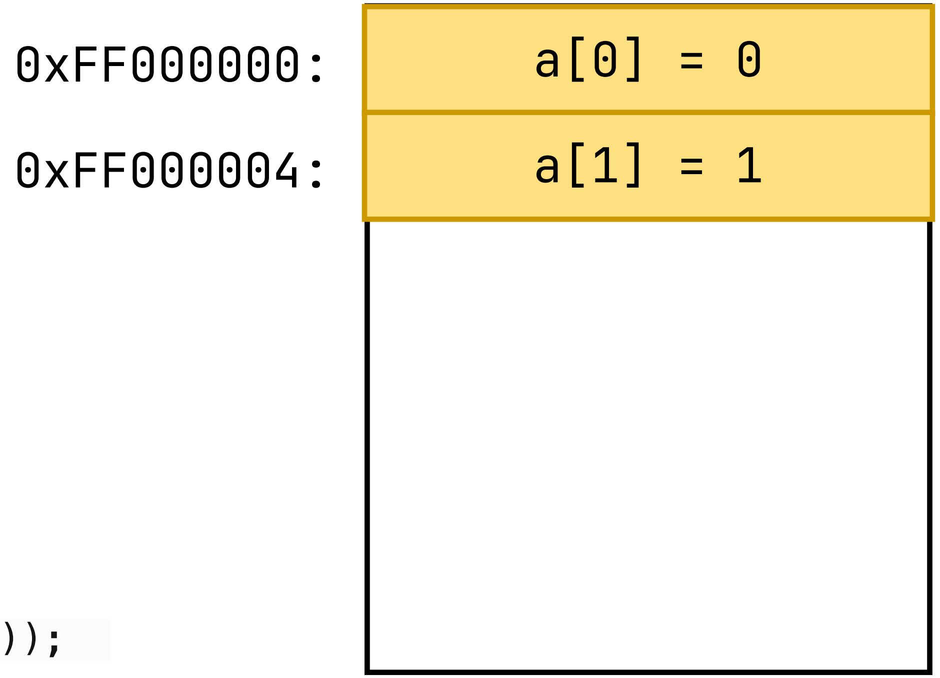
0xFF000000: a[0] = 0

0xFF000004: a[1] = 1

(2)

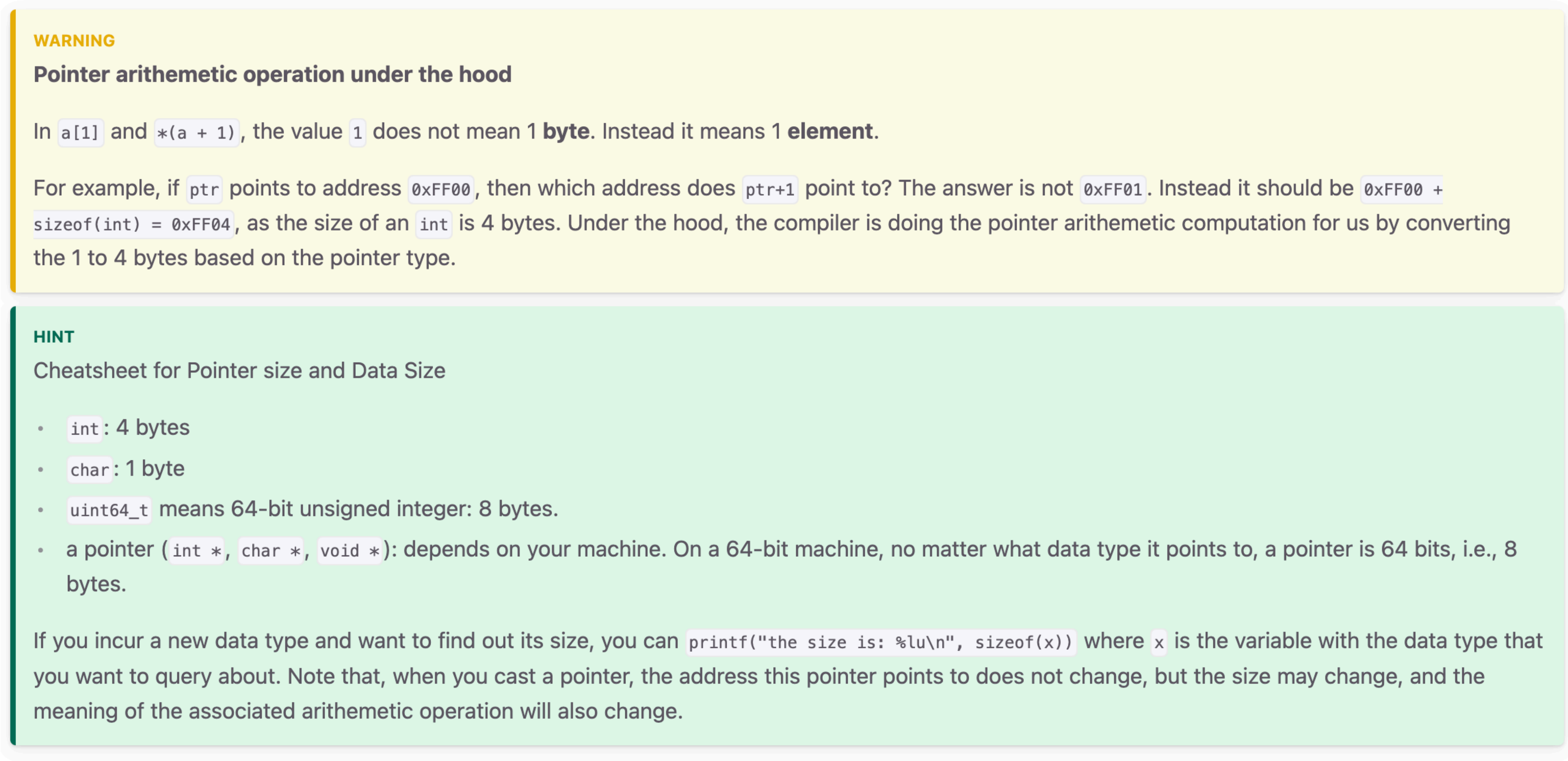
Arrays

```c
void example_method() {
  int a[2];

  a[0] = 0;
  a[1] = 1;

  printf("The first element of a is: %d\n", a[0]);
  printf("The second element of a is: %d\n", a[1]);

  printf("The first element of a is: %d\n", *(a));
  printf("The second element of a is: %d\n", *(a + 1));
}
```



(3)

```
> The first element of a is: 0
> The second element of a is: 1
> The first element of a is: 0
> The second element of a is: 1
```



WARNING

Pointer arithemetic operation under the hood

In `a[1]` and `*(a + 1)`, the value `1` does not mean 1 **byte**. Instead it means 1 **element**.

For example, if `ptr` points to address `0xFF00`, then which address does `ptr+1` point to? The answer is not `0xFF01`. Instead it should be `0xFF00 + sizeof(int) = 0xFF04`, as the size of an `int` is 4 bytes. Under the hood, the compiler is doing the pointer arithemetic computation for us by converting the 1 to 4 bytes based on the pointer type.

HINT

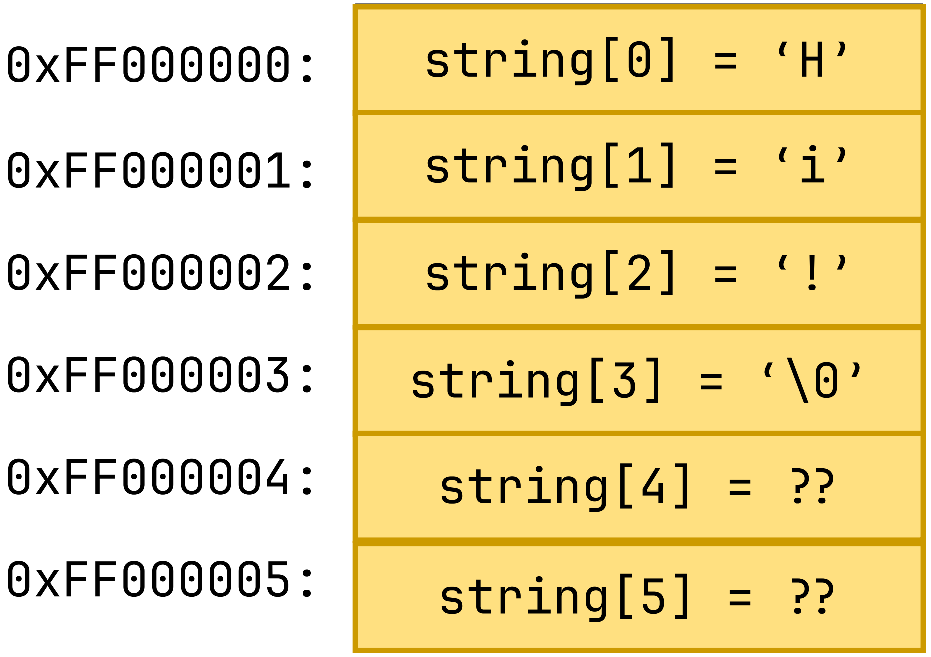
Cheatsheet for Pointer size and Data Size

- `int`: 4 bytes
- `char`: 1 byte
- `uint64_t` means 64-bit unsigned integer: 8 bytes.
- a pointer (`int *`, `char *`, `void *`): depends on your machine. On a 64-bit machine, no matter what data type it points to, a pointer is 64 bits, i.e., 8 bytes.

If you incur a new data type and want to find out its size, you can `printf("the size is: %lu\n", sizeof(x))` where `x` is the variable with the data type that you want to query about. Note that, when you cast a pointer, the address this pointer points to does not change, but the size may change, and the meaning of the associated arithemetic operation will also change.

Strings

```
void example_method() {
  char string[6] = "Hi!";
  printf("String stored: %s\n", string);

  // Print it character-by-character
  int i = 0;
  while(string[i] != '\0') {
    printf("Character %d of string: %c\n", i, string[i]);
    i++;
  }
}
```



0xFF000000:	string[0] = ‘H’
0xFF000001:	string[1] = ‘i’
0xFF000002:	string[2] = ‘!’
0xFF000003:	string[3] = ‘\0’
0xFF000004:	string[4] = ??
0xFF000005:	string[5] = ??

malloc()

```
void example_method() {
  int *array = malloc(2*sizeof(int));
  if (array == NULL) {
    printf("malloc failed! \n");
    return -1;
  }

  array[0] = 1;
  array[1] = 2;

  free(array);
}
```

std::map (C++)

```
std::map<uint64_t, uint64_t> cpp_map;
uint64_t key = 0xDEAD;
uint64_t value = 0xBEEF;

// Add or modify a key-value pair
cpp_map[key] = value;

// Retrieve a value for a key
uint64_t key2 = 0xBAAD;
uint64_t value2 = cpp_map[key2]
if (value2 == 0){
    assert("Key does not exist!\n");
}
```

Conclusion

- Here are some C resources to explore on your own:
  - Learn C in Y Minutes: https://learnxinyminutes.com/c/
  - The C Programming Language (K&R)
  - Style: https://www.kernel.org/doc/Documentation/process/coding-style.rst