

Pointers

Bachelor of Science - École polytechnique

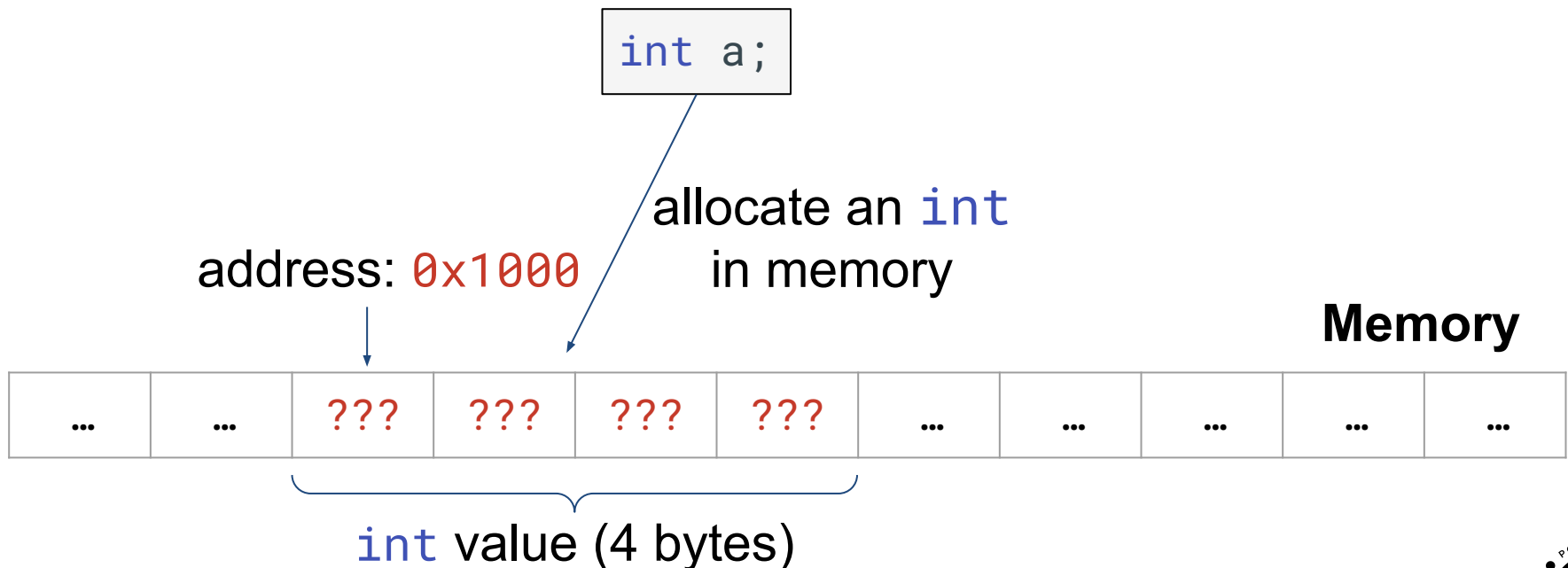
gael.thomas@inria.fr

Key concepts

- Declare a pointer: `type* var ;`
- Get a pointer: `&var`
- Dereference a pointer: `*var`
- Passing an argument by pointer

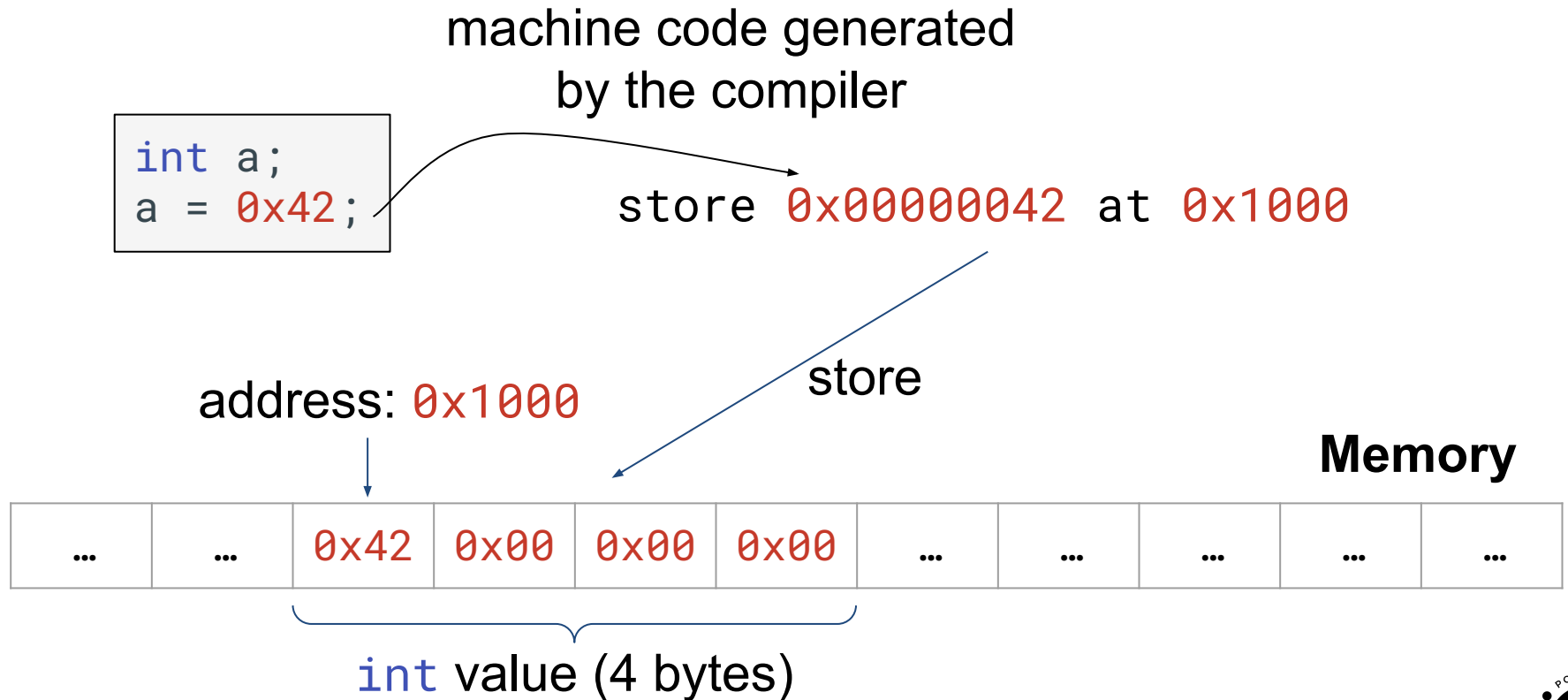
Memory and address (1/2)

- The memory is a big of array of bytes
 - The **index** of a byte in this array is called an **address**
 - Declaring a variable means allocating a memory region
 - Each **variable** has thus an **address**: the **address of its first byte**
 - Here, we say that the address of a is **0x1000**



Memory and address (2/2)

- At runtime, the name of a variable disappears
 - `a` is a symbol that means “address `0x1000`”
 - The machine code replaces each instance of `a` by `0x1000`



Address of a variable

- To retrieve the address of a variable:
 - prefix the variable with `&`
 - and print it by using `"%p"` in `printf`

```
int a = 0x42;  
printf("a is at %p\n", &a);  
//      => a is at 0x1000
```

Pointer type (1/2)

- You can declare **a variable that contains an address**
 - We say that the variable is a **pointer**
- To declare a pointer: **type*** name ;
 - **type**: the type pointed by name
 - *****: indicates that the variable is a pointer
 - **name**: the name of the variable

```
int a = 0x42;  
int* p = &a;  
printf("a is at %p\n", p);  
//      => a is at 0x1000
```

Pointer type (2/2)

- You can declare **a variable that contains an address**
 - We say that the variable is a **pointer**
- And you can access **the value pointed** by a **pointer**
 - We say that we **dereference** the pointer
 - To dereference a pointer `p`: `*p`

```
int a = 0x42;  
int* p = &a;  
  
*p = 0x666; // <=> a = 0x666  
  
printf("a is at %p and its value is 0x%x\n", p, a);  
//    => a is at 0x1000 and its value is 0x666
```

Cool, but what are pointers used for?

- Pointers are used everywhere in C
 - You will discover where in the next lessons
 - For the moment, we will use them to modify a value in a caller
- Sometimes, you want to return multiple values from a function
 - An error code: in C, the convention is 0 if ok, -1 of error
 - And a value
- Example: a function that divide the integer a by the integer b
 - If $b == 0 \Rightarrow$ returns an error
 - Otherwise returns the a/b
 - Problem: any integer can be the result of a/b , we can thus not use the result a/b to indicate if an error occurred
 - We need two variables to return the result

Passing an argument by pointer


p is a pointer used to store the result a / b

```
int div(int* p, int a, int b) {  
    if(b == 0) {  
        return -1  
    } else {  
        *p = a / b;  
        return 0;  
    }  
}  
  
int main(int argc, char* argv[]) {  
    int res;  
    int err = div(&res, 33, 0);  
    if(err != 0) {  
        printf("An error occurred\n");  
    }  
    return 0;  
}
```

Passing an argument by pointer

p is a pointer used to store the result a / b

```
int div(int* p, int a, int b) {  
    if(b == 0) {  
        return -1  
    } else {  
        *p = a / b;  
        return 0;  
    }  
}
```

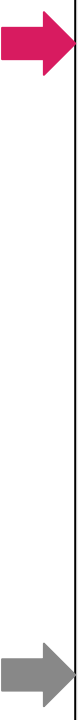


```
int main(int argc, char* argv[]) {  
    int res;  
    int err = div(&res, 33, 0);  
    if(err != 0) {  
        printf("An error occurred\n");  
    }  
    return 0;  
}
```

argc	1
argv	a value
res	
err	
	frame of main

Passing an argument by pointer

p is a pointer used to store the result a / b

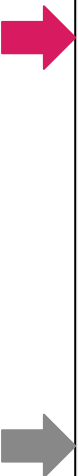


```
int div(int* p, int a, int b) {  
    if(b == 0) {  
        return -1;  
    } else {  
        *p = a / b;  
        return 0;  
    }  
}  
  
int main(int argc, char* argv[]) {  
    int res;  
    int err = div(&res, 33, 3);  
    if(err != 0) {  
        printf("An error occurred\n");  
    }  
    return 0;  
}
```

argc	1
argv	a value
res	
err	
	frame of main
p	0x1000
a	33
b	3
	frame of div

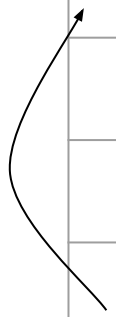
Passing an argument by pointer

p is a pointer used to store the result a / b



```
int div(int* p, int a, int b) {  
    if(b == 0) {  
        return -1;  
    } else {  
        *p = a / b;  
        return 0;  
    }  
}  
  
int main(int argc, char* argv[]) {  
    int res;  
    int err = div(&res, 33, 3);  
    if(err != 0) {  
        printf("An error occurred\n");  
    }  
    return 0;  
}
```

argc	1
argv	a value
res	11
err	
frame of main	
p	0x1000
a	33
b	3
frame of div	



Passing an argument by pointer

p is a pointer used to store the result a / b

```
int div(int* p, int a, int b) {  
    if(b == 0) {  
        return -1  
    } else {  
        *p = a / b;  
        return 0;  
    }  
}  
  
int main(int argc, char* argv[]) {  
    int res;  
    int err = div(&res, 33, 3);  
    if(err != 0) {  
        printf("An error occurred\n");  
    }  
    return 0;  
}
```

argc	1
argv	a value
res	11
err	0
frame of main	

=> two results:

- err is a direct result
- res is an indirect result

Key concepts

- Declare a pointer: `type* var ;`
- Get a pointer: `&var`
- Dereference a pointer: `*var`
- Passing an argument by pointer
- Be careful, two uses of `*`: to declare and dereference a pointer