Today's plan

- Class interactions
  - Ex 2-3
  - Keys points
  - Recap discussion
- Class exercises
  - Ex 3-1

Ex 2-3

Volunteers?

- Get the sense of using command-line arguments with a C program.
- Write a little small function, which is important! Learn to divide your big programs into small pieces and finish it one by one.
- A lot on file I/O and its error handling. You should now know
    - the structure in C representing files: `FILE`. We use a pointer to it.
    - how to use `fopen` and its return value.
    - how to use `fscanf` and its return value.
    - how to use `fprintf` and its return value.
    - how to use `ferror` and its return value.
- In a word, you should know how to read a file, parse the content until the end of the file, and how to write to a file.

Q & A

`https://pigeonhole.at/8ZTP5Q/q/1588952`

Key points - more functions and arrays

- Function declarations v.s. function definitions.
- Preprocessor, compiler, linker.
- Using `sizeof` to get the size of an array.
- Be aware of the difference between array size and string length.
- Passing arrays to functions. What are copied? A pointer, aka. an address.

  `void foo(int a[]); int f[5] = {0}; foo(f);`

Symbols	Values
f[0]	0
f[1]	0
⋮	⋮
a	&f[0]

- We can change the array's values in a function using the address we copied.

Key points - recursion

- In short: a function calling itself with stopping criteria.
- When: a divide and conquer approach - i.e. same logic can be used to a smaller problem.
- Example: compute the nth Fibonacci sequence.
  $f_1 = 0, f_2 = 1, f_n = f_{n-1} + f_{n-2}, n \geq 3$
  - Using for loop:

```c
int fibonacci(int n) {
        assert(n >= 1);
        int f_p = 0, f_c = 1;
        if (n == 1) return f_p;
        if (n == 2) return f_c;
        for (int i = 3; i <= n; ++i) {
                int f_n = f_p + f_c;
                f_p = f_c, f_c = f_n;
        }
        return f_c;
}
```

Key points - recursion

- Example: compute the nth Fibonacci sequence.
  $f_1 = 0, f_2 = 1, f_n = f_{n-1} + f_{n-2}, n \geq 3$
  - Using for recursion:

```
int fibonacci(int n) {
        assert(n >= 1);
        if (n == 1) return 0;
        if (n == 2) return 1;
        return fibonacci(n-1) + fibonacci(n-2);
}
```

- Drawback: causing deep stack, which can crash your program when you are running out of stack spaces.
- Think if a problem can be divided into a similar but smaller problem.
- Can you use recursion to detect if a word is Palindrome? Think about how to divide it into a subproblem.

What is the down-side to recursion?

Mentimeter

limited by memory

large memory usage

may create a large stack

it creates a large stack

large stack

ly

it uses a lot of memory

uses a lot of memory

memory overhead

9

How does passing an integer array to a function differ from passing a single integer variable into a function?

Mentimeter

Passes by address, not value.

The function can change the array, but not the single integer

Array: pass a pointer to its first element. Single integer: passing a copy

functions can alter arrays

array: pass by address; integer: pass by copy

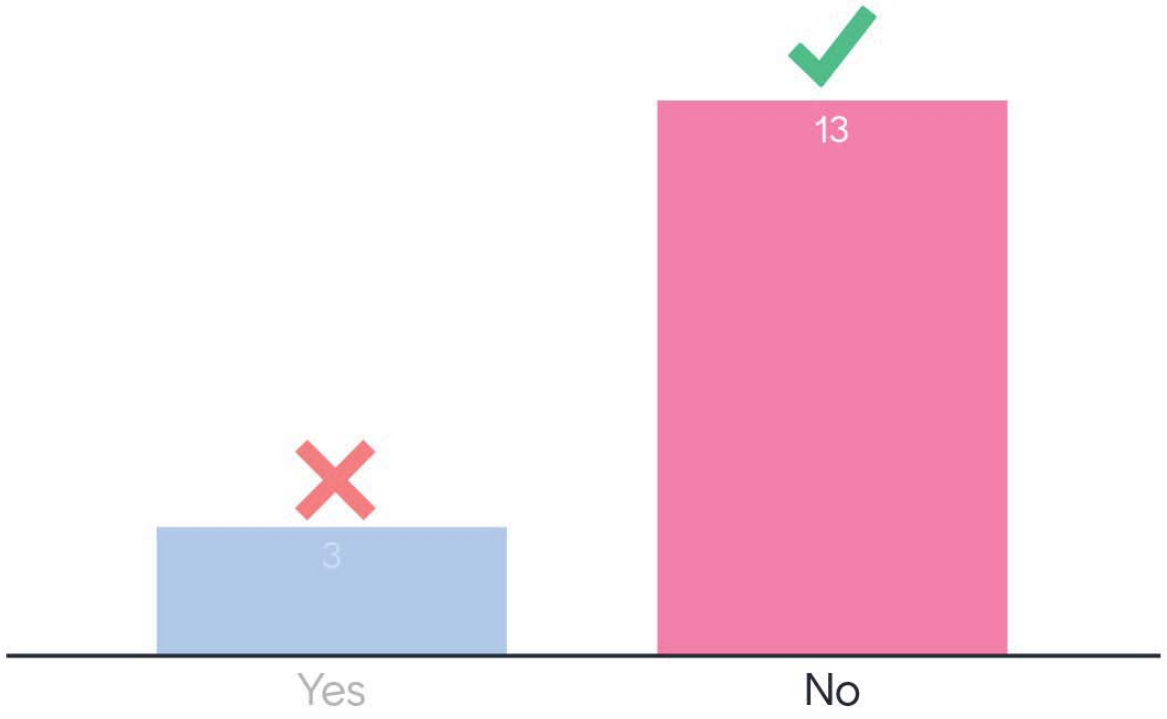
the integer is passed by a value and an array is passed by a pointer

An integer array is passed by pointer and the actual array is modified while when an integer is passed it's passed-by-value and the actual integer isn't modified

7

Can you have two functions with the same function name in a C program?

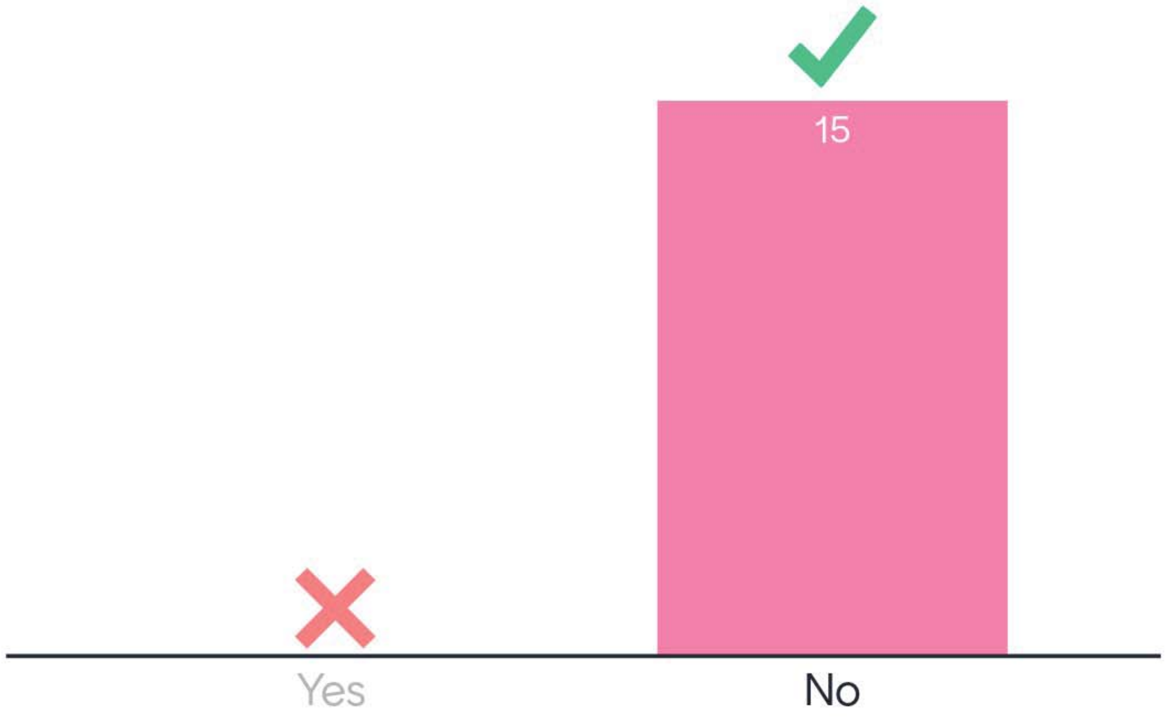
Mentimeter



16

Is the size of a string array the same as the string length?

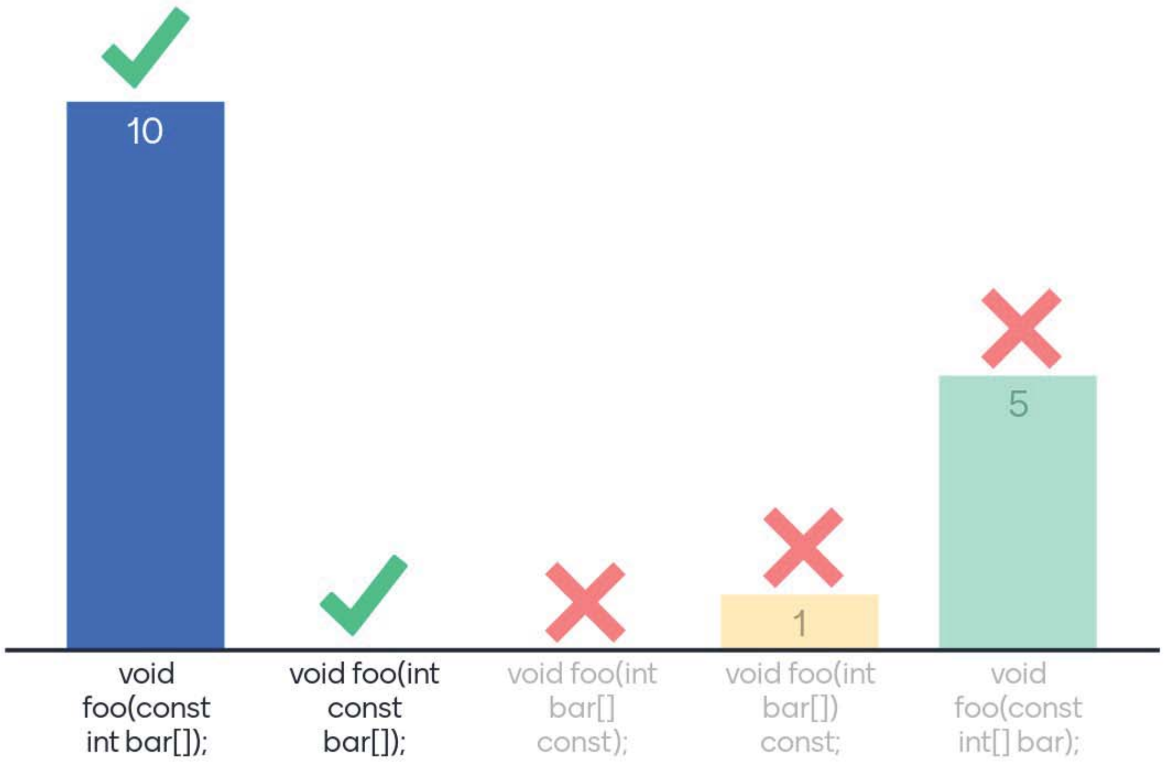
Mentimeter



15

Mentimeter

How can you make an array that is passed into a function not modifiable?



16

Mentimeter

How do you get the size of an integer array? e.g. int a[] = {5,9,10,4,5};

sizeof(a) ✕ 2x	sizeof(a); ✕ 2x	sizeof() ✕
size(a) ✕	sizeof ✕	

The correct answer is: sizeof(a) / sizeof(a[0]);

11

What is the difference between a function declaration and a function definition?

Mentimeter

declaration only needs the header part

function declarations does not require the function to be written. The function can be defined later

Definition is full implementation whereas declaration just states the prototype of the method like function name, output type and types of the input

a declaration lets the compiler know a function exists. A definition actually defines the code that will be ran inside the function.

a function declaration calls the function and uses it and the function definition has the code for the function

a function declaration doesn't have to include the body of the function, but is just created so it can be called from main before it is fully written

Dec

The correct answer is: function declaration has no body (the curly bracket part). A function can be declared multiple times, but we can only define it once.

9

Recap questions

What is the output of the following program?

```c
#include <stdio.h>
void myFunc(int x, int a[]) {
    x += 3;
    a[0] = 42;
}
int main(void) {
    int y = 4;
    int r[] = {1, 2, 3};
    myFunc(y, r);
    printf("y=%d, r[0]=%d\n", y, r[0]);
    return 0;
}
```

At line 10:

Symbols (Scope)	Values
y (main)	4
r[0] (main)	42
r[1] (main)	2
r[2] (main)	3

Answers: `y=4, r[0]=42`

Class exercises

Ex 3-1