C - POINTERS AND FUNCTIONS

RECAP LAST CLASS

- stack vs heap
- array name is actually just pointer
- how to allocate memory

MORE ON MEMORY ALLOCATION

- Two types:
  - compile time memory allocation
    - stack
    - Ex: `int a;`, `int arr[20];`
  - runtime (dynamic) memory allocation
    - heap
    - Ex: anything using `malloc()`, `calloc()`, `realloc()`

WHEN TO USE DYNAMIC (HEAP)?

- Need a lot of memory
  - Stack is typically fairly limited in size
  - Request too much -> program crashes
- Needs to persist after function returns
  - Ex: array made in function that still need access to in main
- If amount of memory needed unknown / changes
  - Ex: array size not determined in advance
  - Ex: array that needs to change size
- Safest to use if size is not known at compile time
  - VLAs (C99) allow, but as of C11 it's optional

FUNCTIONS

Example:

```
double sum(double a, double b) {
    double res;
    res = a+b;
    return res;
}
```

FUNCTIONS (CONT.)

- Can go before or after main
- If after, must put function prototype before
  - `double sum(double a, double b);` or
  - `double sum(double, double);`
- functions that don't return anything have `void` return type
- can only return 1 value
- best to get in habit of using function prototype and defining later
  - will come up again when we look at bigger programs

FUNCTIONS

Pass-by-reference or Pass-by-value?

- C is always pass-by-value
  - sometimes that value is a value
  - sometimes it is a pointer

FUNCTIONS AND POINTERS

- If you want to be able to change the variable:
  - pass a pointer
  - function declaration must specify argument is pointer
- Passing arrays
  - Always passes a pointer
  - Typically pass size as separate argument
- Returning arrays
  - Either return pointer (or modify in place)