Today’s lecture

- **Pointers/arrays**
  - Mechanics, syntax
  - Underlying memory model
  - Array indexing == pointer arithmetic
  - As parameters

- **Stack versus heap allocation**
  - Stack declaration, scope, lifetime
  - Heap allocation/deallocation

CULTURE FACT:
IN CODE, IT’S NOT CONSIDERED RUDE TO POINT.
C type system

Each variable declared with type; determines size of storage and valid operations

Operations required to respect that type

Can’t multiply two char *, can’t deference an int
Co-mingle distinct types accepted if "sensible" automatic conversion exists

Pointer variables distinguished by type of pointee

Dereferencing int* yields an int, dereferencing char* yields a char
Pointer arithmetic on int* scales by sizeof(int), on char* scales by sizeof(char)

Memory is sequence of bytes, no type information

What is stored at address 0x7ffff3460? A char? 4-byte int? Uninitialized bits?
What if you access data at location with incorrect idea of type?
Type system operates at compile-time only, no runtime type information
Array is sequence of elements, homogenous type

```c
int arr[5];
```
Allocates space for 5 ints, contiguous memory, indexed from 0 to 4

Subscript to access individual elements

```c
arr[0] = 72
arr[1] = 45
```

What happens if subscript invalid?

```c
arr[99] = 10
arr[-1] = 3
```

Can assign array to pointer — what does this do?

```c
int *ptr = arr;
```
Use of array name "decays" to address of first element, e.g. `arr` is equivalent to `&arr[0]`
Array contents not copied on assignment, `ptr` assigned address in memory where `arr` stored
ptr and `arr` are now "aliases", refer to same memory
Array indexing is "syntactic sugar" for pointer arithmetic

\[ \begin{align*}
    & \text{ptr + i} \quad \iff \quad &\&\text{ptr}[i] \\
    & *(\text{ptr + i}) \quad \iff \quad &\text{ptr}[i]
\end{align*} \]

Arithmetic scaled by sizeof(pointee)

\[ \begin{align*}
    & \text{ptr + 1 adds one if ptr is char *, adds 4 if ptr is int *} \\
    & \text{What happens if you cast to different size pointee before arithmetic?}
\end{align*} \]

Either syntax on either pointer or array

Can use subscript on pointer variable or pointer arithmetic on array

Access to nth element in either always takes into account size of pointee
Similar…. but not identical
Consider C type system & draw pictures to visualize how underlying reality is same/different

Operations in common
Dereference, pointer arithmetic, array indexing

Difference in declaration
What space is allocated and what does memory diagram look like?
Array declaration set aside space for N elements
Pointer declaration is single variable to hold address

Difference in operations
Can reassign the pointer to hold a different address, not so with array
arr = NULL doesn’t even compile — why not?
What is sizeof(ptr)? what is sizeof(arr)?
Let’s code & draw!

/afs/ir/class/cs107/samples/lect6

arrptr.c
Stack allocation

- **Very efficient**
  - Fast to allocate/deallocate, ok to oversize
- **Not especially plentiful**
  - Total stack size fixed, default 8MB
- **Convenient**
  - Automatic allocation/deallocation on function entry/exit
  - Can declare and initialize in one step
- **Size fixed at declaration, no option to resize**
  - Size can be constant or runtime expression, but once sized, cannot change
  - Stack array cannot be re-assigned -- there is no pointer to array start!
- **Reasonable type safety**
- **Scope/lifetime**
  - Dictated by control flow in/out of functions
Heap allocation

```c
void *malloc(size_t nbytes);
void free(void *ptr);
void *realloc(void *ptr, size_t nbytes);
```

- **void**
  - Variable of type address with unspecified/unknown pointee type

- **What you can do with a void** *
  - Pass to/from function, pointer assignment

- **What you cannot**
  - Cannot dereference
  - Cannot do pointer arithmetic
  - Cannot use array indexing (depends on both arithmetic & dereference!)
Heap allocator analogy

- **Request memory by size**
  - Receive room key to first of connecting rooms

- **Need more room?**
  - Extend into connecting room if available
  - Or trade for new digs, bellman moves your stuff for you

- **Checkout when done**
  - You remember your room number though

- **Errors! What happens if you…**
  - Forget to check out?
  - Bust through connecting door to neighbor?
    - what if neighboring room in use? yikes!
  - Return to room after checkout?
  - Request 3 connecting rooms and only discontiguous avail?