
MONTANA
STATE UNIVERSITY

Pointers & Arrays In C

•••

The Fun Stuff!

Pointers

- A pointer is a variable that contains the **memory address** of another variable
- This is one of the more difficult concepts to understand for people new to C

```
int my_int_function(int *pointer_to_int) {
    return *pointer_to_int;
}
```

Pointers

- A big part of the confusion around pointers is the syntax
- The * character is used in both the declaration of a pointer as well as the *dereferencing* of a pointer

```
int my_int_function(int *pointer_to_int) {
    return *pointer_to_int;
}
```

Pointers

- Look at the function below
- This is a function that takes a pointer to an integer value
- It returns an integer

```
int my_int_function(int *pointer_to_int) {
    return *pointer_to_int;
}
```

Pointers

- In the body of the function, it *dereferences* the pointer to the integer and returns it
- Dereferencing means “get the thing at the address”

```
int my_int_function(int *pointer_to_int) {
    return *pointer_to_int;
}
```

Pointers

- How do we go the other way: I have a thing, how do I get the address of that thing?
- Use the *‘&’* operator

```
int i = 10;
int *pointer_to_i = &i;
```

Pointers

- DEMO #1

```
int i = 10;
int *pointer_to_i = &i;
```

Pointers

- Pointers can be confusing
- So why use pointers?
  - Pointers are small (just an integer)
  - Allow you to pass around references to large pieces of data rather than copies of data
  - Allow you to do low level tricks at times
  - Because that’s the way C works
  - We’ll talk more about why in a bit

Arrays

- Arrays in C are probably more familiar to you
- Declaration is a little funny: brackets come *after* the array variable name
  - Very strange language design choices
  - As with pointers, the type information is bound to the symbol, rather than the type information

```
int int_arr[] = {1, 2, 3};
for (int i = 0; i < 3; ++i) {
    printf( format: "Value: %d", int_arr[i]);
}
```

Arrays

- You can declare an array size as well
- What does this code print out?

```
int int_arr[] = {1, 2, 3};
for (int i = 0; i < 3; ++i) {
    printf( format: "Value: %d", int_arr[i]);
}
```

Arrays

- *“If there are fewer initializers in a brace-enclosed list than there are elements or members of an aggregate, or fewer characters in a string literal used to initialize an array of known size than there are elements in the array, the remainder of the aggregate shall be initialized implicitly the same as objects that have static storage duration.”*
- o_O wat
- Let’s just avoid this usage...

Arrays

- Generally, try to fully, explicitly initialize all data in C
- Here is a syntactic shortcut for initializing the array to 0

```
int int_arr[4] = {0};
for (int i = 0; i < 4; ++i) {
    printf("Value: %d\n", int_arr[i]);
}
```

Pointer & Array Correspondence

- Pointers and arrays are deeply related to one another
- This becomes apparent as you delve into *pointer arithmetic*
- You can add to and subtract from pointers
  - This is known as “pointer arithmetic”
- These operations bump the address one “slot”
  - The size of the slot bump is determined by the size of the pointer type
- A demo will make this clearer
  - *ARRAY POINTER DEMO*

Pointer & Array Correspondence

- So we can see that pointers and arrays are closely related
- Pointers are raw addresses
- Arrays are a starting address of a series of values
  - Array indexing is just pointer math
    - arr[10] → *(arr_ptr + 10)
- You *can* assign an array to a pointer
- You *can't* assign an pointer to an array
- There are structural reasons that pointers are much more common in C than arrays

Array Restrictions

- Arrays are not very commonly used in C
- Why not?
- Restrictions on arrays:
  - You can convert an array to a pointer easily but not vice-versa
  - You can dynamically allocate data as a pointer, but not as an array
  - You can’t return an array from a function (!!!)
- These restrictions are why most C apis use pointers rather than arrays (e.g. char *)
- Arrays are sometimes used as members of structs, however

Pass By Value (Why use pointers?)

- You may still be wondering why we would ever use pointers
- We haven’t discussed C calling conventions yet, so let’s get into it
- C is a *pass by value* language
- This means that all argument values are *copied* into a functions parameters
  - Java and most higher level programming languages also work this way, but Java “hides the pointer”
  - So, what does this mean?

Pass By Value (Why use pointers?)

- Updates to the value of a parameter do not update the original value that was passed in
- Intuitive in the case of “primitive” values
- Arrays work as expected (because they act like pointers)
- What about structs?
  - What’s a struct?

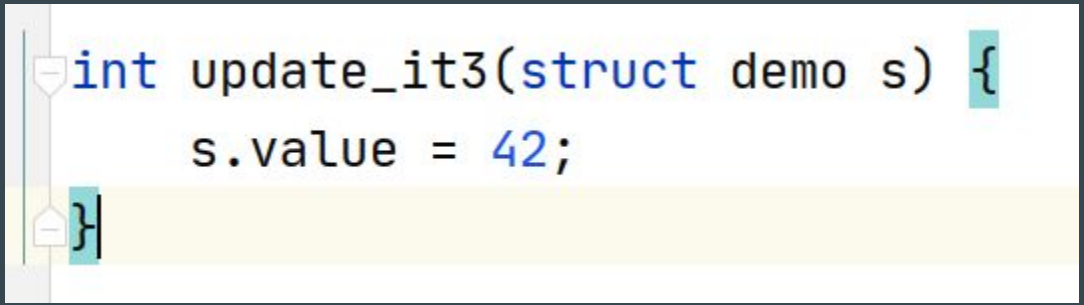
Structs

- We will talk more about structs in a future lecture, but think of them as a collection of data
- Here is a very simple struct that holds a single integer

```
struct demo {
    int value;
};
```

Structs

- Another update function, this time updating the slot in the struct



Structs

- What does this print?

```c
int main() {
    struct demo s;
    s.value = 10;
    update_it3(s);
    printf("Value: %d\n", s.value);
}
```

Structs

- Answer: 10!
- Why?

```
int main() {
    struct demo s;
    s.value = 10;
    update_it3(s);
    printf("Value: %d\n", s.value);
}
```

Structs

- To update this properly, we need to pass a *pointer* to the struct, dereference it, and update the field

```
int update_it4(struct demo *s) {
    (*s).value = 42;
}
```

Pass By Value (Why use pointers?)

- OK, this is a little confusing
- This is why C is a hard programming language to use
  - Idiosyncratic syntax
  - Confusing semantics
    - Why do arrays work the way I expect, but structs don’t?
      - Because arrays act like pointers
        - Mostly
- Playing around with simple examples is the best way to learn
- Don’t give up! It’s just code!

Out Parameters (Why use pointers?)

- Another reason that you will see pointers used frequently is because functions need to return multiple values
- Let's look at the *read* function:

  *int read(int fildes, void *buf, size_t nbyte);*

- The function returns the number of bytes read, but you also want those bytes
- This is often called an "Out" parameter

Out Parameters (Why use pointers?)

- Another reason that you will see pointers used frequently is because functions need to return multiple values
- Let's look at the *read* function:

```
int read(int fildes, void *buf, size_t nbyte);
```

- The function returns the number of bytes read
- But you also want those bytes!

Out Parameters (Why use pointers?)

- Let's look at the *read* function:

    *int read(int fildes, void *buf, int nbyte);*

- To give you the bytes as well, you need to pass in a buffer to be filled out
- You pass in a pointer to this buffer so the function can fill in something you have access to

Final Topic: Pointers to Pointers

- You can probably see, to an extent, the usefulness of pointers
- BUT WAIT, THERE'S MORE
- You will occasionally see a function with a *pointer to a pointer* in it

Final Topic: Pointers to Pointers

- Example: strtok_r, the thread safe string tokenizer function

```c
#include <string.h>
void strtok_example() {
    char str[] = "C is pretty crazy...";
    char *token;
    char *str_ptr = str;
    char **rest_of_str_ptr = &str;

    while ((token = strtok_r(str_ptr, " ", rest_of_str_ptr)))
        printf("%s\n", token);
}
```

Final Topic: Pointers to Pointers

- Pointers to pointers are usually used when a function needs to return more than one value
- In this case it is returning the next token, called token, and also a new pointer to the next spot to tokenize from, which the rest_of_string_ptr is updated to point to
- Recall, C passes BY VALUE
- So you can’t pass just a pointer in: you want to update the pointer
- Therefore, a pointer to a pointer

Final Topic: Pointers to Pointers

Review

- Pointers are addresses of other values
- They are declared using the * character
- Confusingly, they are dereferenced using the * character as well
- Arrays are a bit more intuitive
- Arrays and pointers have a correspondence w/ one another
- C is pass by value, so if you want to update something, you need to pass in a pointer to it


MONTANA
STATE UNIVERSITY