STRUCTS

In C, we have the ability to invent our own data types if you will - data types that the authors of C decades ago just didn't envision or just didn't think were necessary because we can implement them ourselves - similar to Scratch, just as you could create custom puzzle pieces, or in C, you can create custom functions. *

So, in C, can you create your own types of data that go beyond the built in ints and floats and even strings?

You can make, for instance, a PERSON DATA TYPE or a candidate data type in the context of elections or a person data type more generically that might have a name and a number.

So how might we do this?

Well, let me go here and propose that if we want to define a person, wouldn't it be nice if we could have a person data type and then we could have an array called people?

And maybe that array is an only array with (2) things in it, (2) persons in it.

But somehow, those data types, those persons, would have both a name and a number associated with them.

So we don't need two separate arrays!

We need 1 array of persons, a brand new data type.

If we want every person in the world or in this program to have a name and a number, we literaly right out the those 2 data types:

```
string name;
string number;
```

And then we wrap that, those 2 lines of code, with this syntax, which at 1st glance is a little cryptic.

It's a lot of words all of a sudden. But typedef is a new keyword today that defines a new data type.

This is the C key word that lets you create your own data type for the very first time.

TYPEDEF → create your own data type

Struct is another related key word that tells the compiler that this isn't just a simple data type, like an int

or a float renamed or something like that.

```
typedef struct
{
    string name;
    string number;
}
person;
```

It's actually a structure.

It's got some dimensions to it, like 2 things in it or 3 things in it or even 50 things inside of it.

The last word down here is the name that you want to give your data type, and it weirdly goes after the curly braces.

But this is how you invent a data type called person and what this code is implying is that henceforth, the compiler along will know that a person is composed of a name that's a string and a number that's a string.

And you don't have to worry about having multiple arrays now. You can just have an array of people moving forward.

So how can we go about using this?

Why don't we enhance the phone book code a little bit by borrowing some of that new syntax?

1º Let me go to the top of my program above main and define a type that's a structure or a data structure that has a name inside of it and that has a number inside of it. And the name of this new structure again is going to be called person:

```
typedef struct
{
    string name;
    string number;
}
person;
```

2º Inside my code now, let me go ahead and delete this old stuff temporarily...

Let me give myself an array called people of size 2:

```
person people[2];
```

I want to be more pedantic ← spell out what I want in this array of size 2 at location 0, which is the 1st person in an array and I'm going to give that person a name of "Carter".

```
people[0].name = "Carter";
```

↳ And the dot is admittedly a new piece of syntax today too.

* the dot means go inside of that structure and access the variable called name and give it this value "Carter". *

Similarly, if I'm going to give Carter a number, I can go into people[0].number and give that the same thing as before "+1-617-495-1000".

```
people[0].name = "Carter";
people[0].number = "+1-617-495-1000";
```

And then I can do the same for myself here:

```
people[1].name = "David";
people[1].number = "+1-949-468-2750";
```

because again, 2 elements, but we started counting at ZERO.

So now, if I scroll down here to my logic, I don't think this part needs to change too much.

I'm still going to iterate 2 times from i is 0 on up to but NOT through 2.

```
for (int i = 0; i < 2; i++)
```

```
if (strcmp(names[i], "David") == 0)
```

But I think this lined code needs to change.

How should I now refer to the i person's name as I iterate?

What should I compare "David" do this time?

`people[i].name`, because people is the name of the array.

[i] is the i° person that we're iterating over in the current loop first 0, then 1, maybe higher if it had more people...

Then dot is our NEW syntax for going inside of a data structure and accessing a variable therein which in this case is NAME. *

And so I can compare David just as before.

So it's a little more verbose, but now arguably this is a better program because now these people are full fledged data types unto themselves.

There's no more honor system inside of my loop, that this is going to line up because in just a moment, I'm going to fix this one last remnant of the previous version.

↳ If I can call back on you again, what should I change numbers [i] to this time?

So gone is the honor system that just assumes that [i] in this array lines up with bracket [i] in this other array. Now why?

↳ There's only (1) array, it's an array called people. The things it stores are persons. A person has a name and a number.

So even though it's kind of minimal admittedly given that this is a short program and given that this kind of made things look more complicated at 1st glance, we're now laying the foundation for just a better design because you really don't want to screw up how the association of names with numbers because every person's name and number is ENCAPSULATED inside of the same data type.

↳ that's a term of art in CS } "Encapsulation means to encapsulate - that is, contain related pieces of information." *

Important !! :)

And thus, we have a person that encapsulates 2 other data types → name
↳ number

And this is just into the foundation for all of the cool stuff we've talked about and you use it every day.

What is an image? Is a bunch of pixels or dots on the screen.

° Everyone of those dots has RGB values associated

with red, green and blue.

You could imagine raw creating a structure in C probably where maybe you have 3 values, 3 variables — I called red, I called green, I called blue.

And then you could name the thing not person but pixel.

Now you could store in C 3 different colors — some amount of red, some green, some blue - and collectively treat it as the color of a pixel.

And you could imagine doing something similar perhaps for video or music.

Music, might have 3 variables - one for the musical note; the duration; the loudness of it.

→ You can imagine coming up with your own data type for music as well. It's a little low level, but we're just using like a familiar contacts application.

We now have the way in code to express MOST any data type that we might want to implement or discuss ultimately.

The purposes for which one to use arrays but use them more responsibly now in a better design but

also to lay the foundation for implementing cooler and cooler stuff per our week zero discussion?

* What's the difference between this and an object in an object oriented language?

Slight side note, C is NOT object-oriented.

Languages like Java and C++ and others which you might have heard of, programmed yourself, had friends program in, are object oriented languages, in those languages they have things called classes or objects which are interrelated.

↓

Can store NOT just data, like variables.
They can also store functions and you kind of sort of do this in C, but it's not sort of conventional.

In C you have data structures that store data.

Java, C++, you have objects that store data and functions TOGETHER.

Python is an object-oriented language as well, so we'll see this issue in a few weeks, but let me

"wave my hands at it for now"!

- Can you use this struct to redefine how an int is defined?

Yes. We talked a couple of times now about integer overflow. And most recently, you might have seen me mention the bug in iOS and MacOS that was literally related to an int overflow. *

That's the result of ints only storing 4 bytes or 32 bits or even as long as 64 bits or 8 bytes, but it's finite.

But if you want to implement some financial software or some scientific or mathematical software that allows you to count way bigger than a typical int or a long, you could imagine John coming up with your own structure and, in fact, in some languages there is a structure called "big int", which allows you to express even bigger numbers.

How? Maybe you store inside of a big int an array of values.

And you somehow allow yourself to store MORE and MORE bits, based on how high you want to be able to

meant.

So now we have the ability now to do most anything we want in the language even if it's not built in for us. ☺

→ Could you define a name and a number in the same line?

Sort of. It starts to get syntactically a little messy, so I did it a little more pedantic line by line.

SQL → database language → storing things in actual databases.

You could not store any of the syntax of phone number, for example, or that punctuation inside of an int. You could store just numbers.

↳ So one motivation for using a string is just I can store whatever the human wanted me to store, including parentheses and so forth.

↳ Another reason for storing things as strings, even if they look like numbers, is in the context of zip codes, in the USA.

" Years ago, actually, I was using Microsoft Outlook.

for my email client and eventually I switched to gmail and outlook at the time lets you export all of your contacts as a CSV file

↓

Comma Separated Values

More on that in the weeks to come too. And that just means I could download a text file with all of my friends and family and their numbers inside of it.

Unfortunately, I open that same CSV file with Excel at the time just to kind of spot check it and see if what's in there was what I expected.

→ Command + Ctrl + S to save it

Excel at least has this habit of sort of reformatting your data. If things look like numbers, it treats them as numbers. And Apple Numbers does this too.

↓

~ Google Spreadsheets does this to nowadays ~

" I then imported my mildly saved CSV file into Gmail and now 10+ years later, I'm still occasionally finding friends and family members whose zip codes are in Cambridge, Massachusetts, 2138, which is missing the 0 because we here in Cambridge are 02138. And that's because I treated

a F let Excel treat what looks like a number as an actual number or int, and now leading zeros become a problem because mathematically, they mean nothing, but in the mail system, they do rendering envelopes and such..."

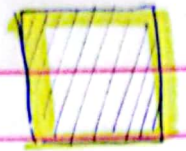
SORTING

So now that we have this ability to STORE different types of data like contacts in a phone book, having names and addresses, let's actually take a step back and consider how we might now solve one of the original problems by actually sorting the information we're given in advance and considering, per our discussion earlier, just how costly, how time consuming is that because that might tip the scales in favor of sorting, then searching, or maybe just NOT sorting and only SEARCHING. *

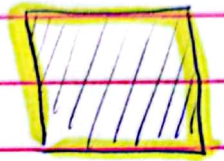
It'll give us a sense of just HOW expensive, so to speak, sorting something actually is.

What's the formulation of this problem?
It's the same thing as week zero.

input → [ ] → output

unsorted →  → sorted

So, if we got numbers like these:

6 3 8 5 2 7 4 1 →  →

if we've got numbers like these, which are just randomly arranged numbers ⟶

we want to get back out 1 2 3 4 5 6 7 8.

So we just want those things to be SORTED.

And, again, inside of the black box here is going to be one or more algorithms that actually gets this job done.

— So how might we go about doing this? —

" Just to vary things a bit more, I think we have a chance here for a bit more audience participation... "