~ 11 ~

— String Comparison —

Maybe this time, I'll store a bunch of names and not just integers but actual strings of names.

So how might I do this? ☺

Well, let me go back to my code here...

I'm going to switch us over to maybe a file called names.c.

```
#include <cs50.h>
#include <stdio.h>
#include <string.h>
```

And I'm going ahead and for now include a new friend from the last week, which gives me some string-related ~~functionality~~ functionality.

So, int main(void) because I'm not going to bother with any command line arguments for now.

```
int main(void)
{
    string names[7];
    names[0] = "Bill";
    names[1] = "Charlie";
```

And now, if I want an array of strings, I could do something like this – string names bracket [7]

↓

And then I could start doing like before...

~~But~~ there's this new improvement I can make.

Let me just let the compiler figure out how many names there are... And using curly braces, I'll do Bill and then Charlie and then Fred and then George and then Ginny and ... if there's the pattern there.

```
{
    string names[] = {"Bill", "Charlie", "Fred", "George", "Ginny",
                      "Percy", "Ron"};
```

So now we have these seven names as strings. Let's do something similar.

For int, i get 0, i is less than 7 as before, i++ !

```
for (int i = 0; i < 7; i++)
{
```

And inside of the loop, let's this time check for the string in question, and suppose we're searching for Ron arbitrarily.

He is there, so we should eventually find him.

Let me go ahead and say if names bracket i equals "Ron", then inside of my if condition, I'm going to say printf found just like before, and I'm going to return 0 just because all is well; And I'm going to take your advice from the get go this time and, at the END of the loop, print out not found because if I get this far, I have not printed found, and I have not returned already.

```
for ( int i = 0; i < 7; i++)
{
      if (names [i] == "Ron")
      {
            printf ("Found\n");
            return 0;
      }
}
printf ("Not found\n");
return 1;
}
```

↳ ERROR → "Result of comparison against a string literal is unspecified. Use an explicit string comparison function instead."

That's kind of a mouthful and the ← 1º time you see it, you're probably not going to know HOW to make sense of that.

But it does kind of draw our attention to something being awry with the equality checking here, with equal equals and Ron.

And there's where again we've been telling sort of a ~~[illegible]~~ lie for the past couple of weeks.

Strings are a thing in C.
Strings are a thing in programming

But recall from last week, I did disclaim there's no such thing as a string data type technically, because it's not a primitive in the way an int, float, bool are, that are sort of built into the language.

You can't just use equation equals to compare 2 strings!
You actually have to use a special function that's in this header file we talked briefly about last week.

#include <string.h>
string length
or strlen
But there's other functions instead as well.

... Let me go ahead and open up the manual pages.
in <manual.cs50.io> *

If we go to string.h, you can perhaps infer what function will probably take the place of == for today.

STRCMP → compares 2 strings

And if I click on that, we'll see more information...

And indeed, if I click on strcmp, we'll see under the synopsis that, ok, I need to use the CS50 header file and string.h, as I already have.

Here is the prototype, which is telling me that strcmp takes 2 strings, s1 and s2, that are presumably going to be compared.

```
int strcmp(string s1, string s2);
```

And it returns an integer, which is interesting.

The description of this function is that it compares 2 strings case sensitively.
So uppercase or lowercase matters, just FYI. And then let's look at the return value here.

The return value of this function returns an int less than 0 if S1 comes before S2, 0 if S1 is the same as S2, or an int greater than 0 if S1 comes AFTER S2.

→ So the reason that this function returns an integer and NOT just a bool, true or false, is that it actually will allow you to sort these things eventually because if you can tell me if 2 strings come in this order or in this order or they're the same, you need 3 possible return values. *

bool gives you 2

But int gives you like 4 billion even though we just need the 3.

So zero
Or positive number
Or negative number
} Is what this function returns.

And the documentation goes on to explain what we mean by ASCIIbetical order.

A is 65
B is 66

And it's those underlying ASCII or Unicode numbers that a computer uses to figure out whether something comes before it or after it like in the dictionary.

But for our purposes now, we only care about equality.

→ So I'm going to go ahead and do this:
If I want to compare names bracket i against "Ron", I use strcmp ~~[illegible]~~ (names [i], "Ron")

if (strcmp (names [i], "Ron")

So it's a little more involved than actually using == , which does work for integers, longs, and certain other values. But for strings, it turns out we need to use a more powerful function.

Why? Well, last week, recall what a string really is.

It's an array of characters ↓

↳ And so whereas you can use ~~$~~ == for single characters, strcmp, as we'll eventually see, is going to compare multiple characters for us.

↳ there's a LOOP needed, and that's why it comes with the string library. But it doesn't just work out of the box with == alone. that would literally be comparing 2 things, not 2 arrays of things.

So I want to check if the return value of strcmp is ==

Digitalizada com CamScanner

to zero, because per the documentation, that means they're the same.

(s1 == s2) → zero.

"You should be checking for 0 or a ⊕ value or ⊖ value."

~~ " ~~