Storing data in Arrays

Suppose that we want to write a program that maybe implements something a little more like a phonebook that has both names and numbers and not just integers but actual phone numbers.

Well, we could isolate things like this: we could now have 2 arrays — one called names, one called numbers, and I'm going to use strings for the numbers now, because in most communities, phone numbers might have dashes, pluses, parentheses, so something that really looks more like a string even though we call it a phone number.

Probably don't want to use an int ~~desk~~ lest we throw away those kinds of details.

So let me switch back to BS Code here, and let's do

1 more program, this one in a file called phonebook.c.

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

int main(void)
{
    string names[] = {"Carter", "David"};
```

And then, inside of my program, I'm going to give myself 2 arrays the efficient way this time.

If I want this to be an array, I don't have to specify the number. The compiler can count for me.

But I do need the square brackets then for numbers, I'm again going to use a string array specifying with the curly braces and that now about Carter can be at 1-617-495-1000.

```
    string numbers[] = {"+1-617-495-1000",
                        "+1-949-468-2750"};
```

If you want to have a little fun with programming, feel free to text or call me sometime at that number.

So now let's actually use this data in some way. Let's go ahead and actually search for my own name and number here ... So:

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

There's 2 of us this time → so I less than 2 and then i++ as before.

And now I'm going to practice what I preached earlier, and I'm going to use strcompare to find my name in this case.

↳ I'm going to say if strcmp of names bracket i equals "David" and that equals 0, meaning they're the same, then just as before, I'm going to go ahead and print something out.

```
for(int i=0; i<2; i++)
{
    if(strcmp(names[i], "David") == 0)
    {
        printf("Found %s\n", numbers[i]);
        return 0;
    }
}
printf("Not found\n");
return 1;
}
```

But this time, I'm going to make the program more useful and not just say found or not found.

↳ I'm implementing a phone book, like the contacts app on iOS or Android.

So I'm going to say something like "Found %s\n", numbers[i]);

So if I do ./phonebook

$ ./phonebook

Found +1-949-468-2750

Ok, that's correct. But there's no ability to add names or edit things.

That, of course, could come later using get string or something else. But for now, for the sake of discussion, I've just hardcoded 2 names and 2 numbers.

↓

But for what it does, I claim this is correct.

↳ It's going to find me and print out my number.

But is it well-designed?

Let's start to now consider if we're not just using arrays, but are we using them, well?

* And what might I even mean by using an array well or designing this program well?

↳ It seems all too vulnerable to just mistakes.

For instance, if I screw up the actual number of names in the names array such that it's now more or less than is in the number array or vice versa, it feels

like there's not a tight relationship between these pieces of data, and it's just sort of is trusting on the honor system that any time I use names[i] that it lines up with numbers[i]. And that's fine.

↓

If you're the one writing the code, you're probably not going to really screw this up. But if you start collaborating with someone else or the program is getting much, much longer, the odds that you or your colleagues remember that you're sort of just trusting that names and numbers line up like this is going to fail eventually.

↳ Someone's not going to realize that, and just, the code is going to break. And you're going to start out putting the wrong number for names, which is to say it'd be much nicer if we could somehow couple these 2 pieces of data, names and numbers, a little more tightly together so that you're not just trusting that these 2 independent variables, names and numbers, have this kind of relationship with themselves.

↓

So let's consider HOW we might solve this.

A new feature today that we'll introduce is generally known as a data structure.