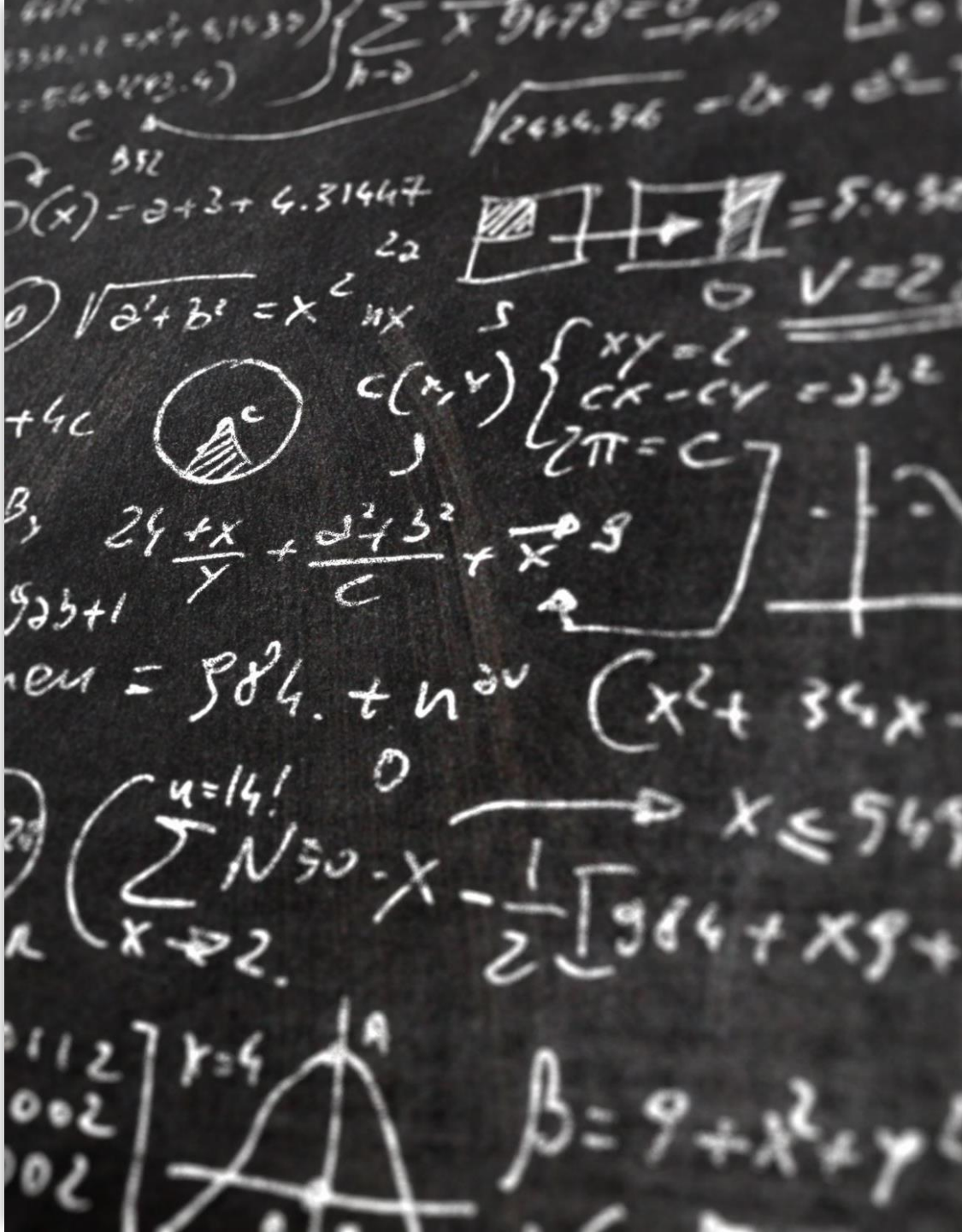
School of Computing & Communications | Lancaster University

SCC.111 Software Development – Lecture 11: Pointers and Strings

Adrian Friday, Nigel Davies, Hansi Hettiarachchi, Saad Ezzini

This lecture

- Consolidating **pointers** and **indirection**
- Introducing **strings** (at last!)
- How **pointers and arrays relate** and help us with strings



Quick quiz on pointers

Join at menti.com | use code 5862 0835

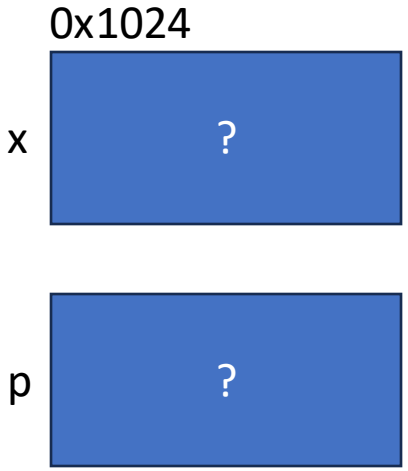
Some examples of pointers

```
int main()
{
  int x; // declares space for an int

  int *p; // declares space for a pointer to an int

  x = 7; // "puts 7 into the box called x"

  p = &x; // "puts the address of box x into p"

  *p = 8; // "puts 8 into the box pointed to by p"

  printf("%d, %p.\n", x, p);
}
```

%p takes a pointer value and prints that address nicely

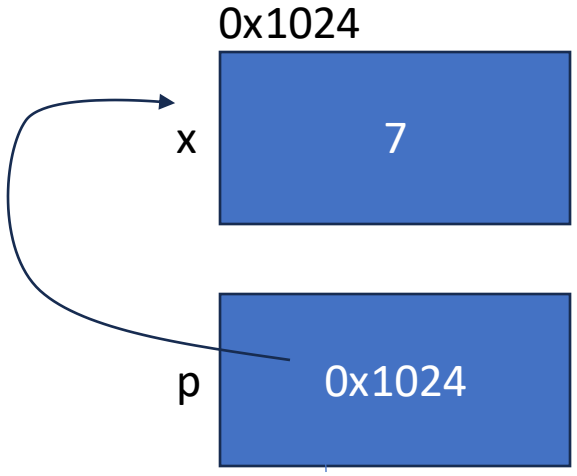
Some examples of pointers

```
int main()
{
  int x; // declares space for an int

  int *p; // declares space for a pointer to an int

  x = 7; // "puts 7 into the box called x"

  p = &x; // "puts the address of box x into p"

  *p = 8; // "puts 8 into the box pointed to by p"

  printf("%d, %p.\n", x, p);
}
```



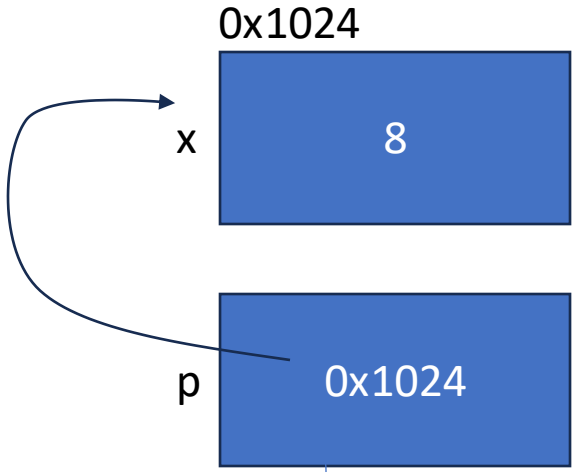
Some examples of pointers

```
int main()
{
  int x; // declares space for an int

  int *p; // declares space for a pointer to an int

  x = 7; // "puts 7 into the box called x"

  p = &x; // "puts the address of box x into p"

  *p = 8; // "puts 8 into the box pointed to by p"

  printf("%d, %p.\n", x, p);
}
```



Some examples of pointers

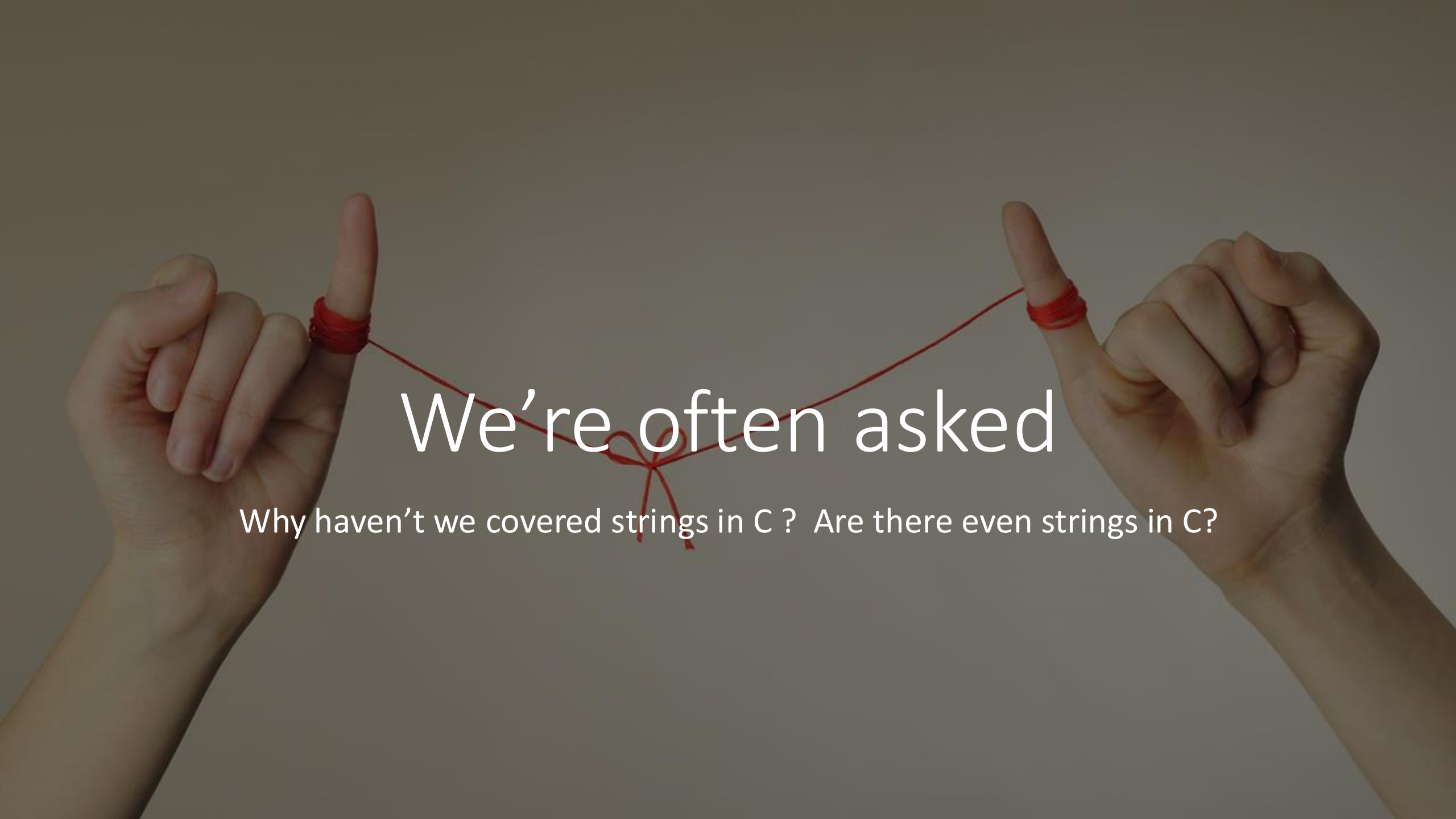
```
int main()
{
  int x; // declares space for an int

  int *p; // declares space for a pointer to an int

  x = 7; // "puts 7 into the box called x"

  p = &x; // "puts the address of box x into p"

  *p = 8; // "puts 8 into the box pointed to by p"

  printf("%d, %p.\n", x, p);
}
```



Recall: Indirection in C

- A **pointer** is a variable that typically contains '*the address of*' another variable
- This allows us to access and change it from elsewhere in our programs
- *But we can use it for all sorts of data manipulation tasks, as we'll see!*

(K&R, ch. 5)



We’re often asked

Why haven’t we covered strings in C ? Are there even strings in C?

Representing strings (textual data) is really important

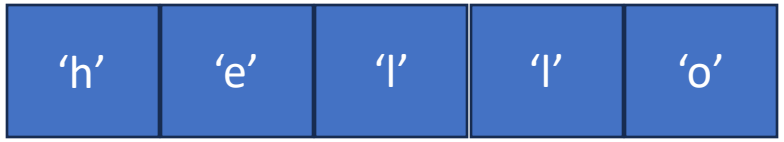
But there are several questions for the designer of the language…

Designing the perfect string

- How long is a string?
- Is there a maximum size?
- How much memory do you set aside to represent one?
- How do you mark how long the string is in memory?
- If strings are ‘any length’, how do you keep your code efficient?

A string is just a sequence of characters…

"hello"



Really ASCII character codes representing letters e.g. ‘h’ is 104

Strings in C

- Strings are not first class data types in C (like **int** or **float**)
- They are really just character (**char**) arrays
- As with arrays, they have a **fixed** maximum **length**
- And can and *often* are *indexed like arrays* (from 0)
- Strings need ‘terminating’ with an *end of string* marker (hidden character **\0**, the ASCII **NUL** character)

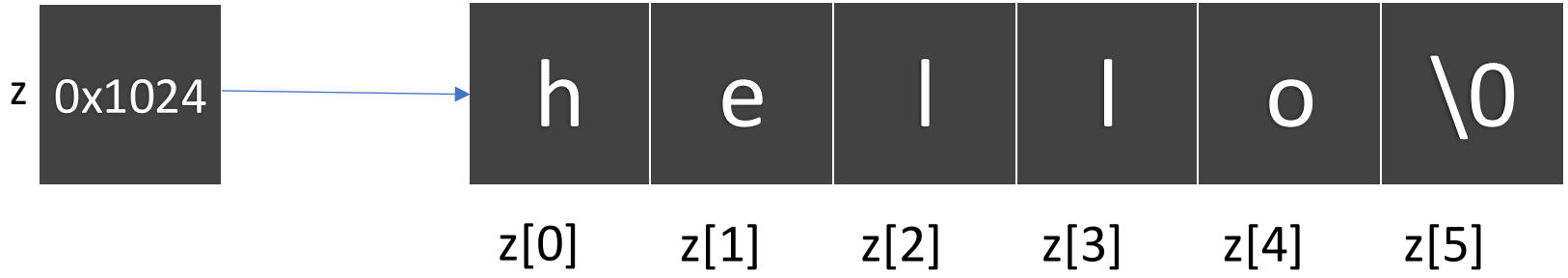
Note how the \0 is added implicitly

```
char z[] = "hello";
```

h	e	l	l	o	\0
z[0]	z[1]	z[2]	z[3]	z[4]	z[5]

z[0] is ‘h’
z[1] is ‘e’
z[5] is zero ‘\0’
really 6 consecutive chars in memory!

```
char *z = "hello";
```

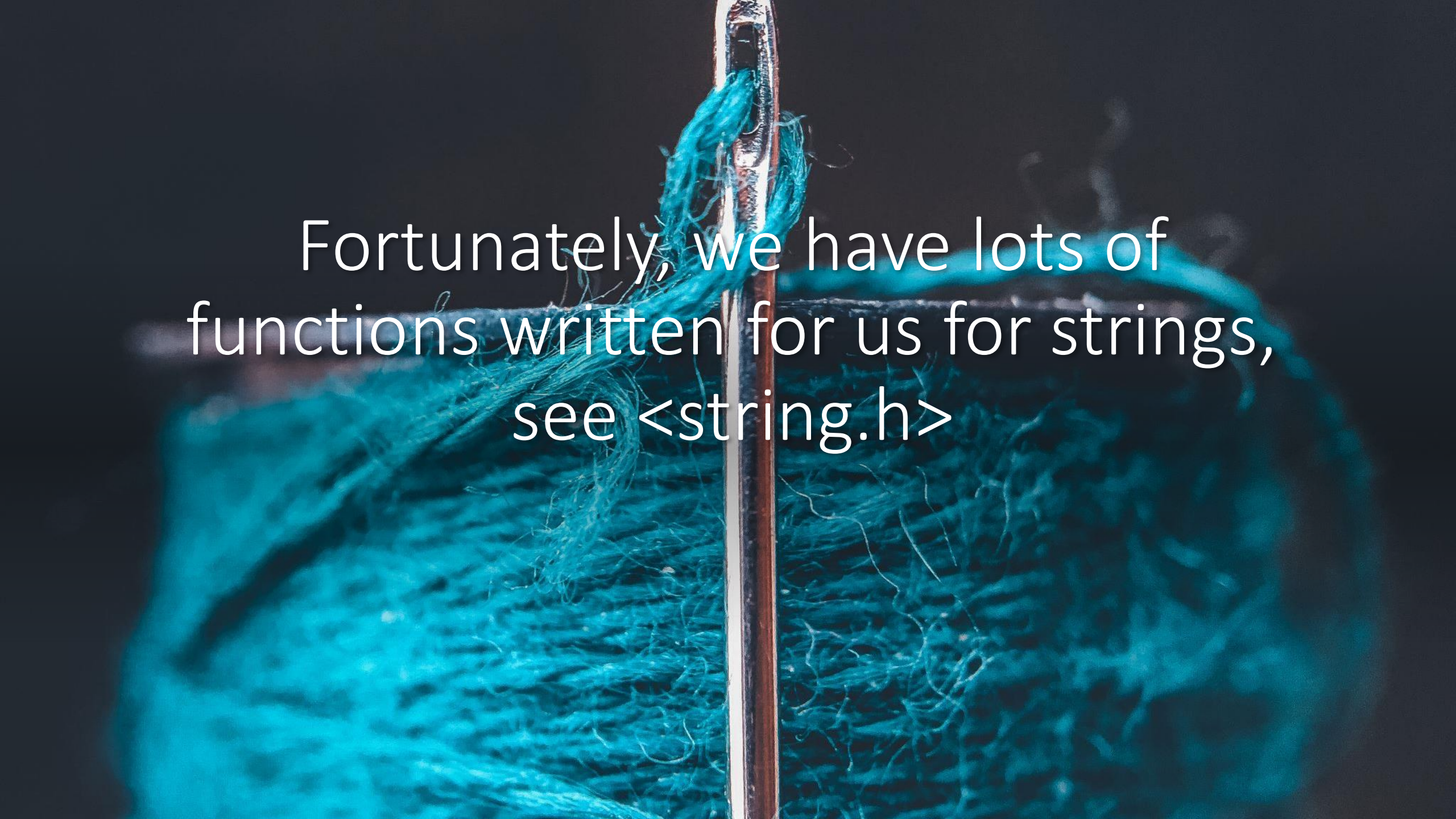


z[0] is ‘h’

*z is ‘h’ too!

Walkthrough: our own function for calculating string lengths (using pointers)

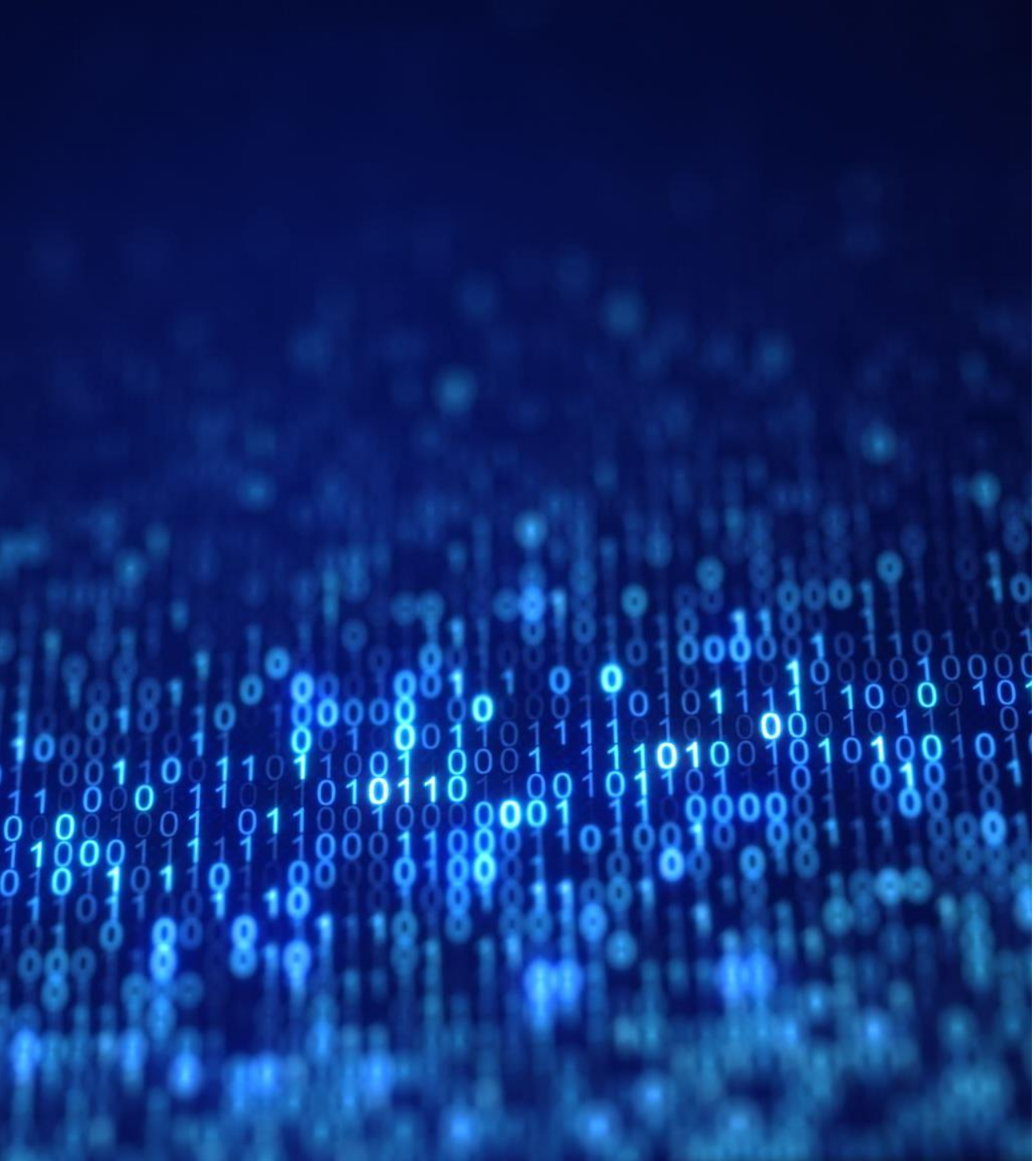
What happens if we miss off the terminating \0 character?



Fortunately, we have lots of functions written for us for strings, see <string.h>

Remember

- There *are no strings*, just arrays of ‘char’s’
- Chars *are* 8-bit integers (the ASCII codes), so non-ASCII characters can’t be used in this way)
- A string must terminate (finish) with a \0 (the NUL) (or zero)
- We can printf() strings using %s
- *Always* allow enough space for the string *plus* its NUL



Summary

- Strings in C (as arrays of characters or char * pointers)
- Why you always need to think about the length of a string carefully when allocating storage
- *Next lecture: more powerful uses of pointers!*