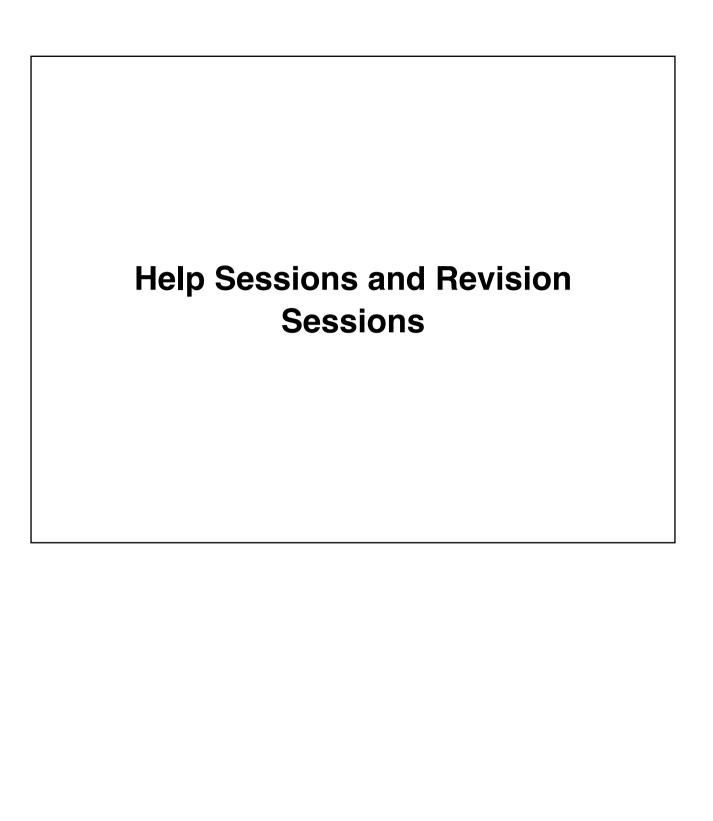
Strings Or, arrays Pt 2



Assignment 1

- Releasing soon
- Watch the Assignment
 Walkthrough live stream
- Watch the Catchup-up video
- Submission in Week 7?
- Worth 20%

Arrays recap

- A collection of data, all of the same type. (homogonous)
- We have a single identifier for the entire array
- It is a random access data structure, meaning we can access any element in the array at any time

The array declaration syntax

int ice_cream_per_day[7];

index:

0 1 2 3 4

5

6

values:

Declare + initialise

```
int ice_cream_per_day[7] = {3, 2,
1, 2, 1, 3, 5};
```

^ Note you can only do this when you declare, not later!

```
int ice_cream_per_day[7] = {};
```

^ Will initialise all elements to 0

Some corrections

```
int my_data[] = {3, 2, 1, 2, 1, 3,
5};
```

^ Will create a 7-element array

```
int my_data[14] = {3, 2, 1, 2, 1,
3, 5};
```

^ Will create a 14-element array, with the first 7 elements then 7 0'd out

Accessing elements

```
int first_day_ice_creams =
ice_cream_per_day[0];
```

index:

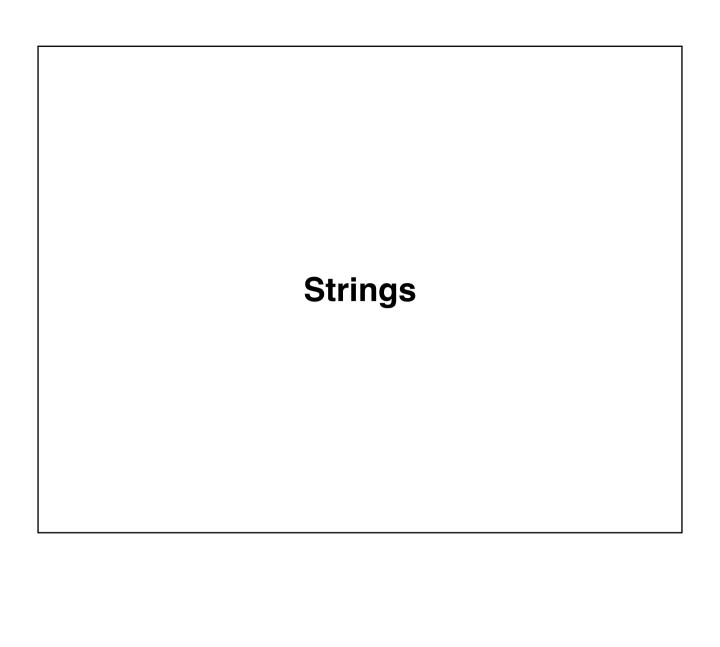
values:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 3 | 2 | 1 | 2 | 1 | 3 | 4 |

Will retrieve the int 3

Writing elements

index: values:



Strings!

- Strings are multi-character words
- "Jake Renzella" -> is a string with 13 characters!
- Strings are great! They are everywhere!

An int array

```
int numbers[7] = {3, 2, 1, 2, 1,
3, 4}
```

index:

values:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 3 | 2 | 1 | 2 | 1 | 3 | 4 |

A char array

index:

O 1 2 3 4 5 6 7 8 9 10 11 12 13

Values: J A K E R E N Z E L L A \0

We can build our own string type by using an array of chars!

Strings in C are char arrays

- A collection of characters
- C does know how to work with

```
char[] S
```

```
#include <stdio.h>

int main(void) {
    char name[3] = {'G', 'a',
    'b'};
    // change name to Jake
    // :( can't, won't fit
    return 0;
}
```

```
#include <stdio.h>
#define MAX_STR 50

int main(void) {
    char name[MAX_STR] = {'J',
    'a', 'k', 'e'};

    return 0;
}
```

New problem

How does C know where the string ends?

```
char name[MAX_STR] = {'J',
'a', 'k', 'e'};
```

The null terminator

- Remember in C, we don't know when arrays end
- We have to keep track of the length ourselves
- We can't always do this with char[] ...





Notice the \ 0 at the end! This means that C will know when it reaches the end of the array

How to use strings in C

- Because strings are character
 arrays, the type is char[]
- There are two ways to declare a string, here's one:

```
char word[] = {'h', 'e', 'l',
'l', 'o', '\0'};
```

| Anyone think that's annoying? |
|-------------------------------|
| |

Strings are very common

So there are easier ways to use them:

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

- This is exactly the same as the previous example
- It includes the null terminator!

String literals

```
"Jake!"
```

- uses double quotes " to wrap the string literal
- single quote for characters!
- Used to assign strings to char[] easily:

```
char name[] = "Jake Renzella";
```

Using strings

- printing: printf or fputs
- scanning: fgets
- Both included in <stdio.h>

fgets

- Reads a string from the terminal
- fgets(array[], length, stream)
 - array[] -> The array that the string will be stored
 - length -> The number of characters that can be read in
 - stream -> The origin of the string (we always use stdin)

fgets usage

```
// Declare the array which will
contain the string. Note, we don't
know how big the string will be, so
let's come up with a maximum.
char my_string[MAX_LENGTH]

// read the string in
fgets(my_string, MAX_LENGTH,
stdin);
```

Reading strings in a loop

- We can read until CTRL+D is entered in the terminal by calling fgets in a loop
- fgets () stops reading when either length-1 characters are read, newline character is read or an end of file is reached, whichever comes first

Reading strings in a loop

```
#include <stdio.h>

// I know my string will never need to be more than 15
chars
#define MAX_LENGTH 15

int main(void) {
    char name[MAX_LENGTH];
    printf("Enter your name: ");

    // fgets reads the entire string, including the newline character
    while (fgets(name, MAX_LENGTH, stdin) != NULL) {
        // every time this runs, we update `name`!
    }
}
```

Printing strings

```
fputs(array[], stream)
```

- array[] -> the character array to be printed
- stream -> the location to print, always use stdout in COMP1511

You can printf a string with %s, but there are security problems with this approach, so we avoid it and use fputs

Printing strings

```
char name[] = "Jake"
fputs(name, stdout)
```

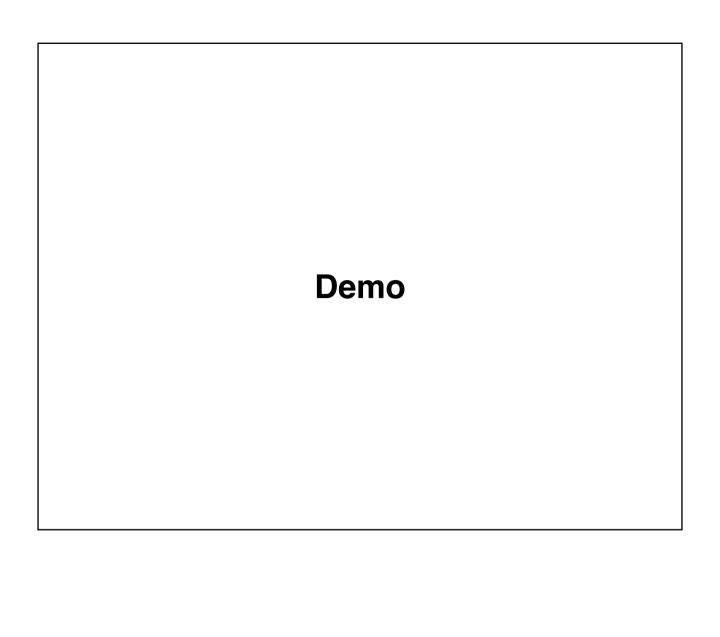
^ Why doesn't fputs need the LENGTH, like fget ?

Other useful string functions

- strlen() -> gives us the length of the string (excluding the \0).
- strcpy() -> copy the contents of one string to another
- strcat() -> join one string to the end of another (concatenate)
- strcmp() -> compare two strings
- strchr() -> find the first occurrence of a character

note: some of these may require | #include

<string.h>



Feedback

https://forms.office.com/r/K3PjvWebtD

