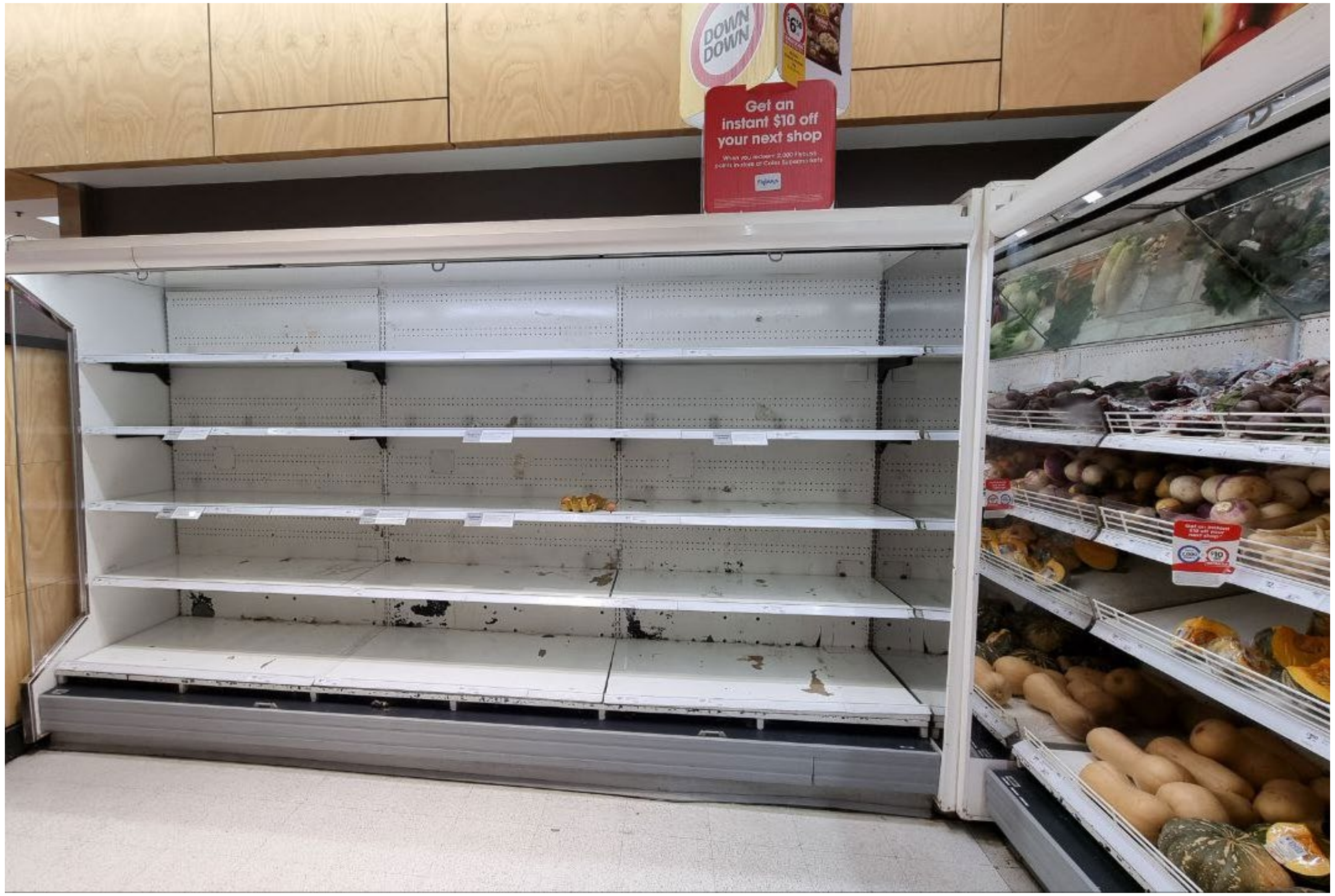


pictures
I took
recently

COMP1511/COMP1911 Week 7!

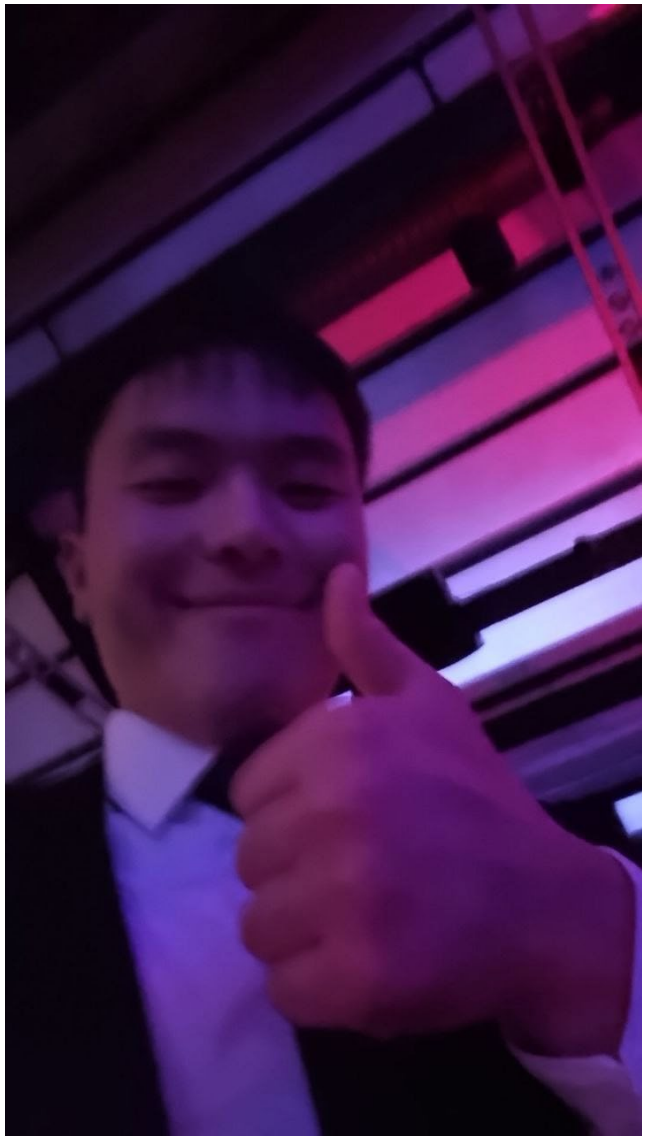
M13B: 1pm – 4pm || T11X: 11am – 2pm

Tutors: William (me!) + Jason || Daniel

My GitHub:

https://github.com/william-o-s/unsw_comp1511_tutoring

Well done on Assignment 1!



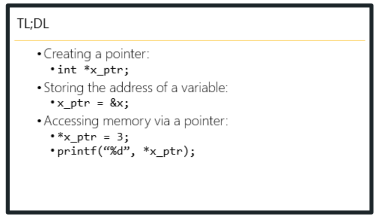
Tutorial Agenda:

Part 1

Part 2

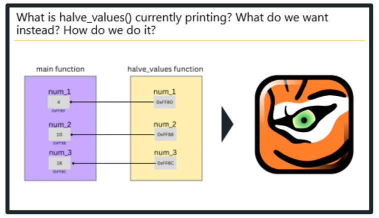
Part 3

Part 4

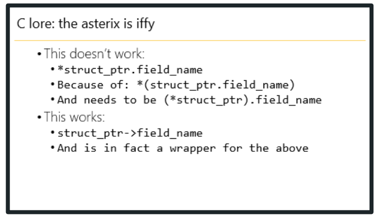


TL;DL

- Creating a pointer:
  - `int *x_ptr;`
- Storing the address of a variable:
  - `x_ptr = &x;`
- Accessing memory via a pointer:
  - `*x_ptr = 3;`
  - `printf("%d", *x_ptr);`

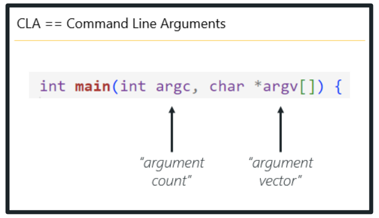


What is halve_values() currently printing? What do we want instead? How do we do it?



C lore: the asterix is iffy

- This doesn't work:
  - `*struct_ptr.field_name`
  - `Because of: *(struct_ptr.field_name)`
  - `And needs to be (*struct_ptr).field_name`
- This works:
  - `struct_ptr->field_name`
  - `And is in fact a wrapper for the above`



CLA == Command Line Arguments

```
int main(int argc, char *argv[]) {
```

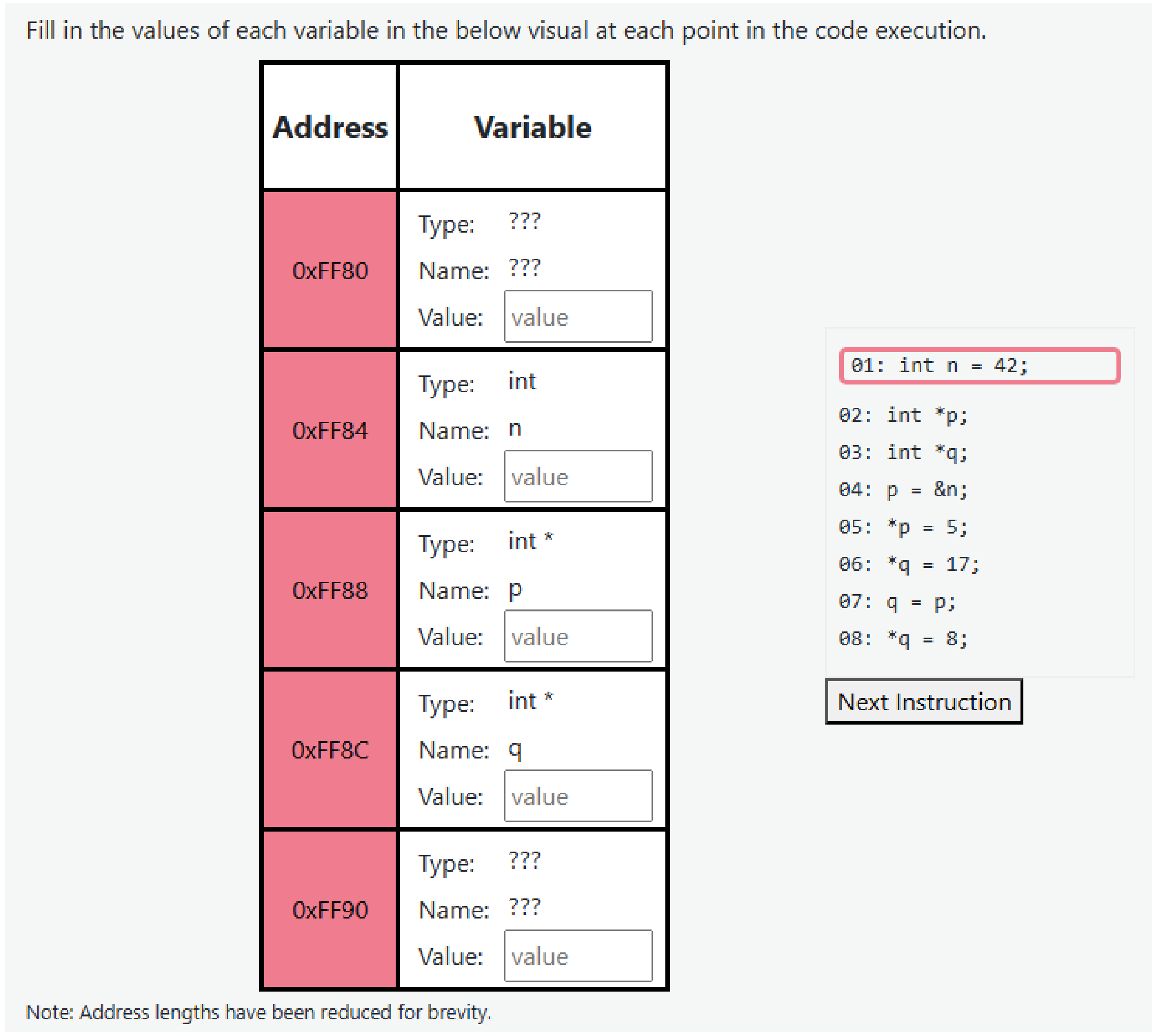
"argument count"

"argument vector"

TL;DL

- Creating a pointer:
  - `int *x_ptr;`
- Storing the address of a variable:
  - `x_ptr = &x;`
- Accessing memory via a pointer:
  - `*x_ptr = 3;`
  - `printf(“%d”, *x_ptr);`

Let's try this cool plugin on the tutorial page (try it with me)



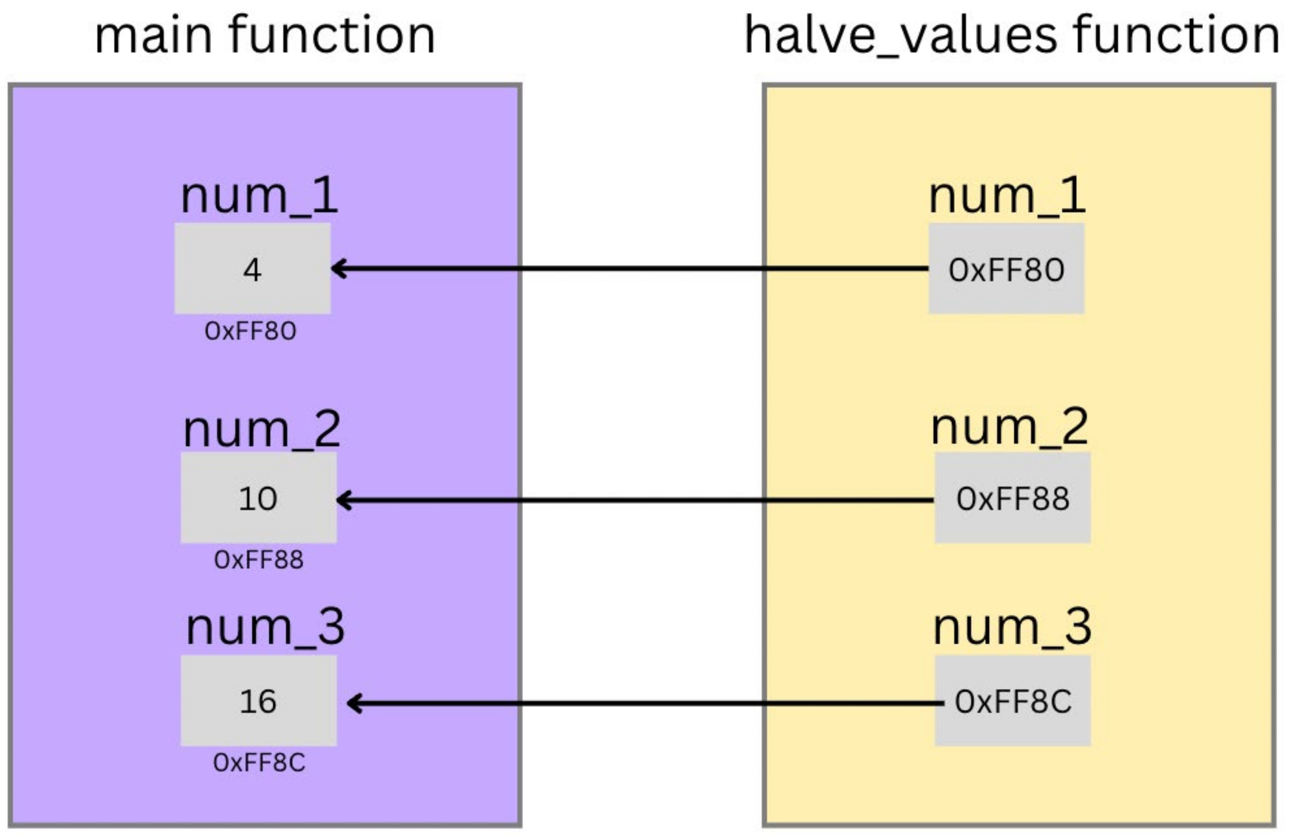
Fill in the values of each variable in the below visual at each point in the code execution.

Address	Variable
0xFF80	Type: ??? Name: ??? Value: value
0xFF84	Type: int Name: n Value: value
0xFF88	Type: int * Name: p Value: value
0xFF8C	Type: int * Name: q Value: value
0xFF90	Type: ??? Name: ??? Value: value

```
01: int n = 42;
02: int *p;
03: int *q;
04: p = &n;
05: *p = 5;
06: *q = 17;
07: q = p;
08: *q = 8;
```

Note: Address lengths have been reduced for brevity.

What is halve_values() currently printing? What do we want instead? How do we do it?

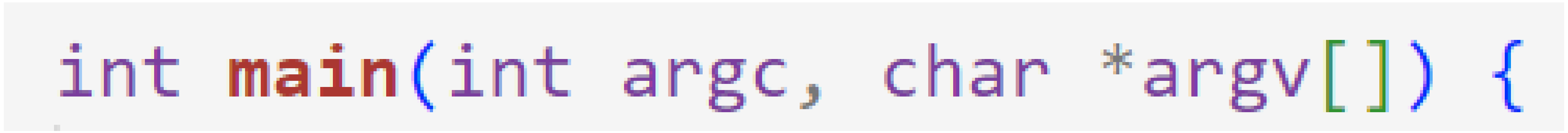


C lore: the asterix is iffy

- This doesn’t work:
  - `*struct_ptr.field_name`
  - `Because of: *(struct_ptr.field_name)`
  - `And needs to be (*struct_ptr).field_name`
- This works:
  - `struct_ptr->field_name`
  - `And is in fact a wrapper for the above`

Now, let's modify a `struct book`!

CLA == Command Line Arguments



```
int main(int argc, char *argv[]) {
```

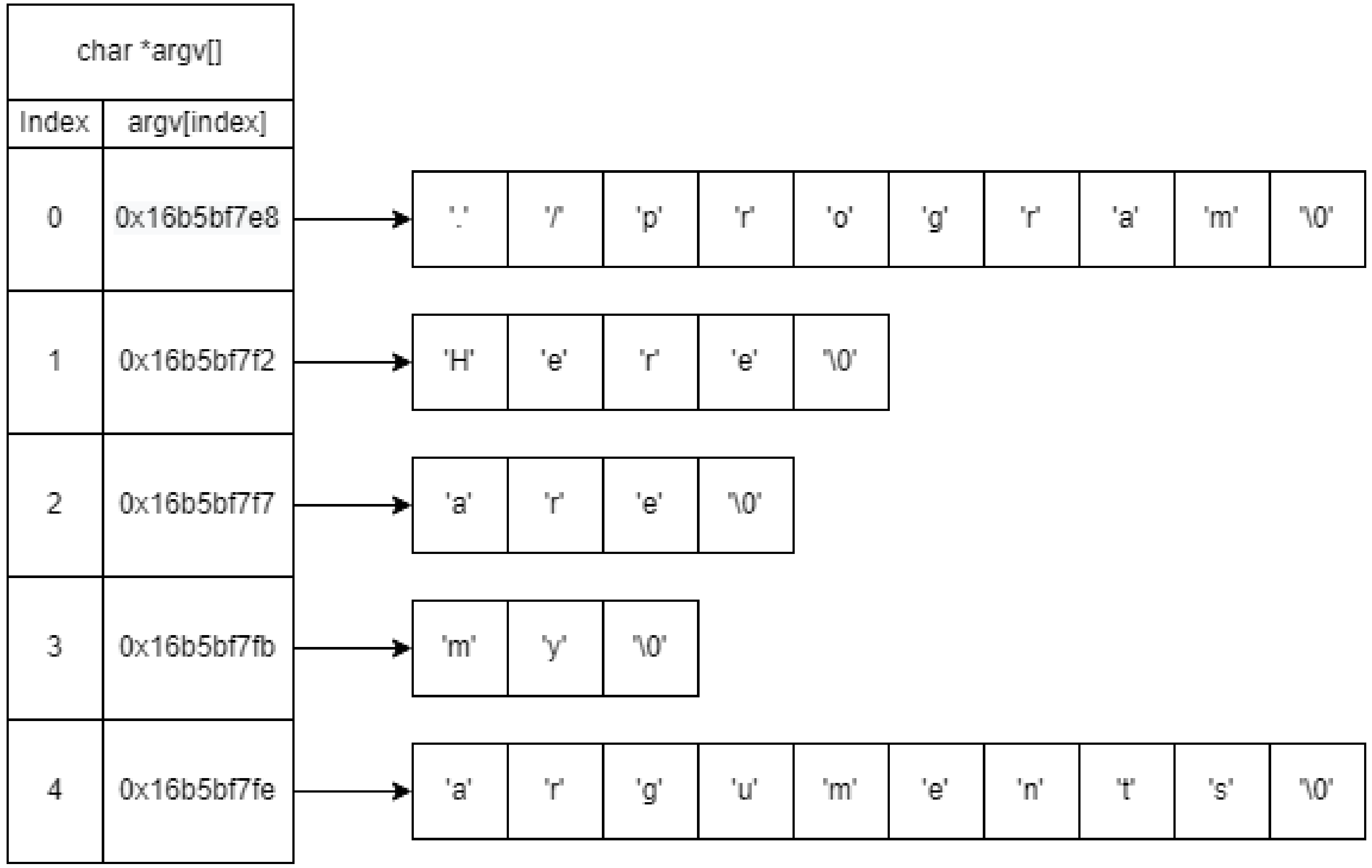
"argument count"

"argument vector"

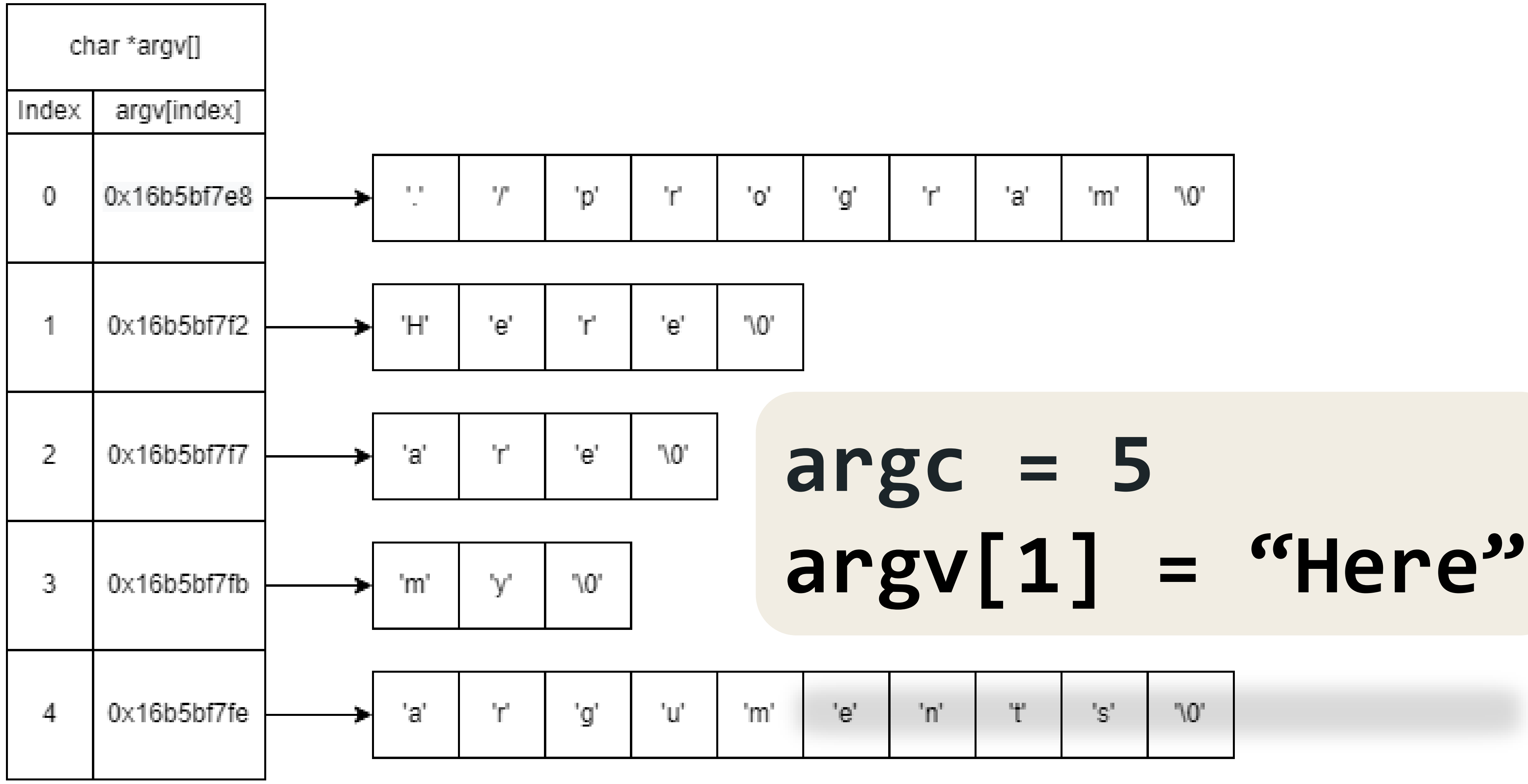
Can you explain what they are?

"argument count"	:	**`argc`** stands for argument count and it represents the number of command line arguments passed to the program, including the name of the program itself
"argument vector"	:	**`argv`** stands for argument vector and it is an array of strings (**`char *`**) that holds the actual command line arguments. The first element (**`argv[0]`**) is always the name of the program, and subsequent elements (**`argv[1]`**, **`argv[2]`**, and so on) hold the additional arguments.

This might help with visualising



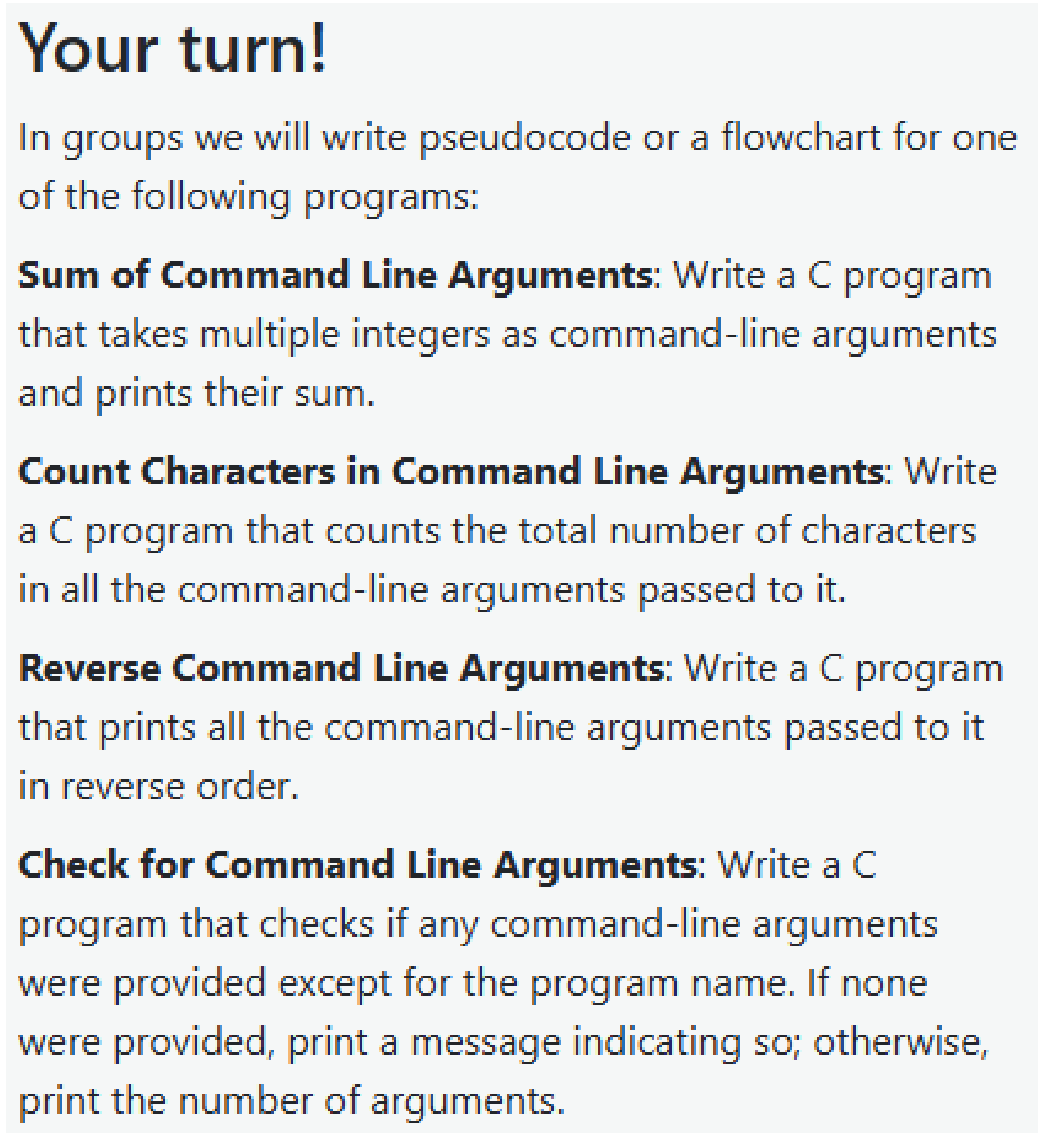
This might help with visualising



```
argc = 5
argv[1] = “Here”
```

Let’s tutor demo a program that counts and prints CLAs

Now let's work on some activities



Your turn!

In groups we will write pseudocode or a flowchart for one of the following programs:

Sum of Command Line Arguments: Write a C program that takes multiple integers as command-line arguments and prints their sum.

Count Characters in Command Line Arguments: Write a C program that counts the total number of characters in all the command-line arguments passed to it.

Reverse Command Line Arguments: Write a C program that prints all the command-line arguments passed to it in reverse order.

Check for Command Line Arguments: Write a C program that checks if any command-line arguments were provided except for the program name. If none were provided, print a message indicating so; otherwise, print the number of arguments.

VSCode Shortcuts

- Start with Ctrl+Shift+P
  - "Toggle Multi-Cursor Editor"
  - Convert text casing: (highlight text) → Ctrl + Shift + P → "Transform to …"
- Multiple Cursors: Ctrl + Click anywhere
  - Cursor over multiple lines vertically: Shift + Alt + Click on line
- Duplicate Line: Ctrl + Shift + Alt + Up/Down Arrow
- Move Lines: Alt + Up/Down Arrow
- Change All Occurrences: Ctrl + Shift + L or Ctrl + D
- Indentation: (Highlight line/lines) → Ctrl + Left/Right Square Bracket
- Find and Replace: Ctrl + F → (click dropdown) → Replace next