# comp1511 week 07

Catherine Cheng

## admin

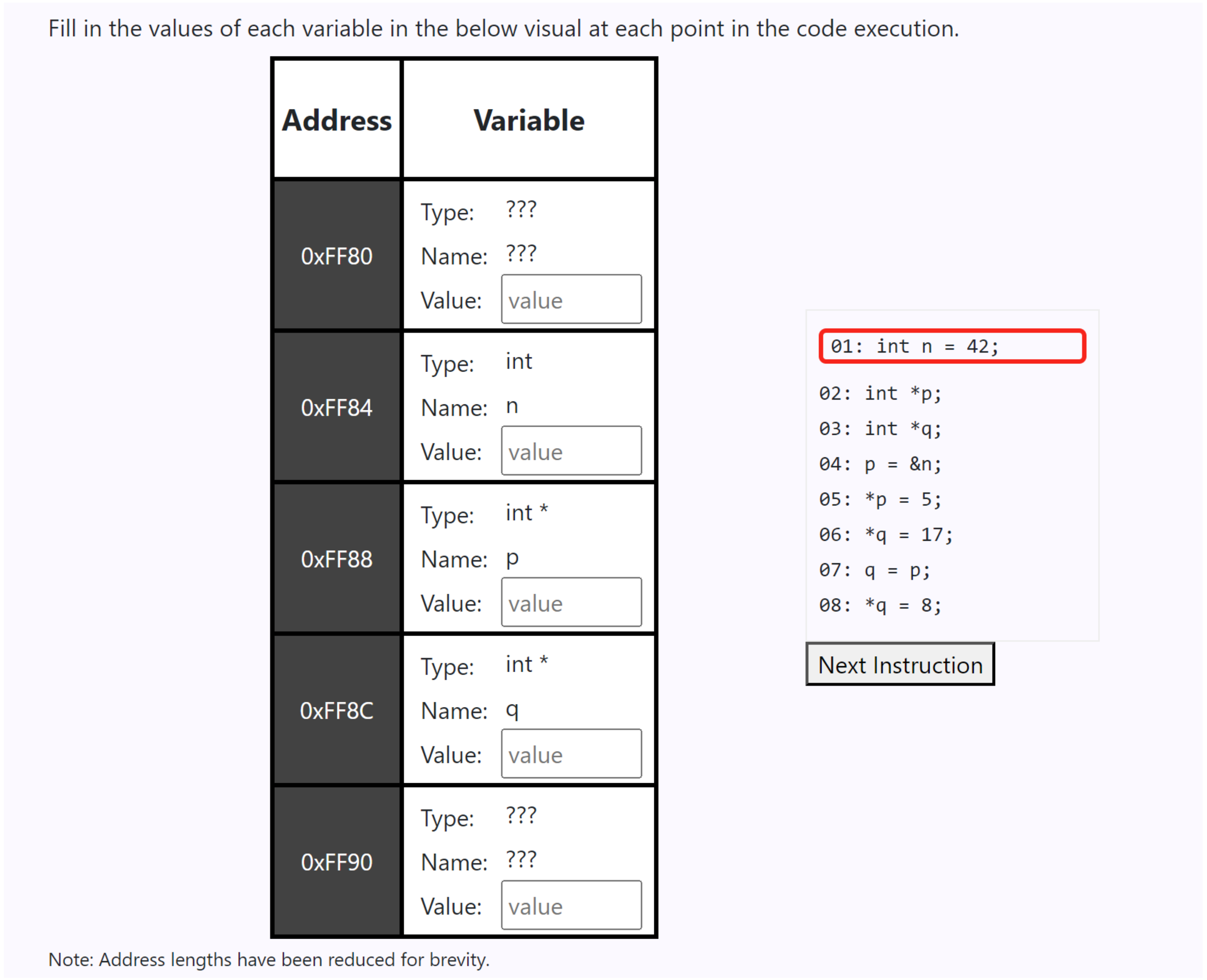
- congrats on finishing **assignment 1!**
- what did we learn?

## agenda for today

- intro to pointers
- using pointers in functions
- struct pointers
- command line arguments

## pointers demo

# pointers example



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

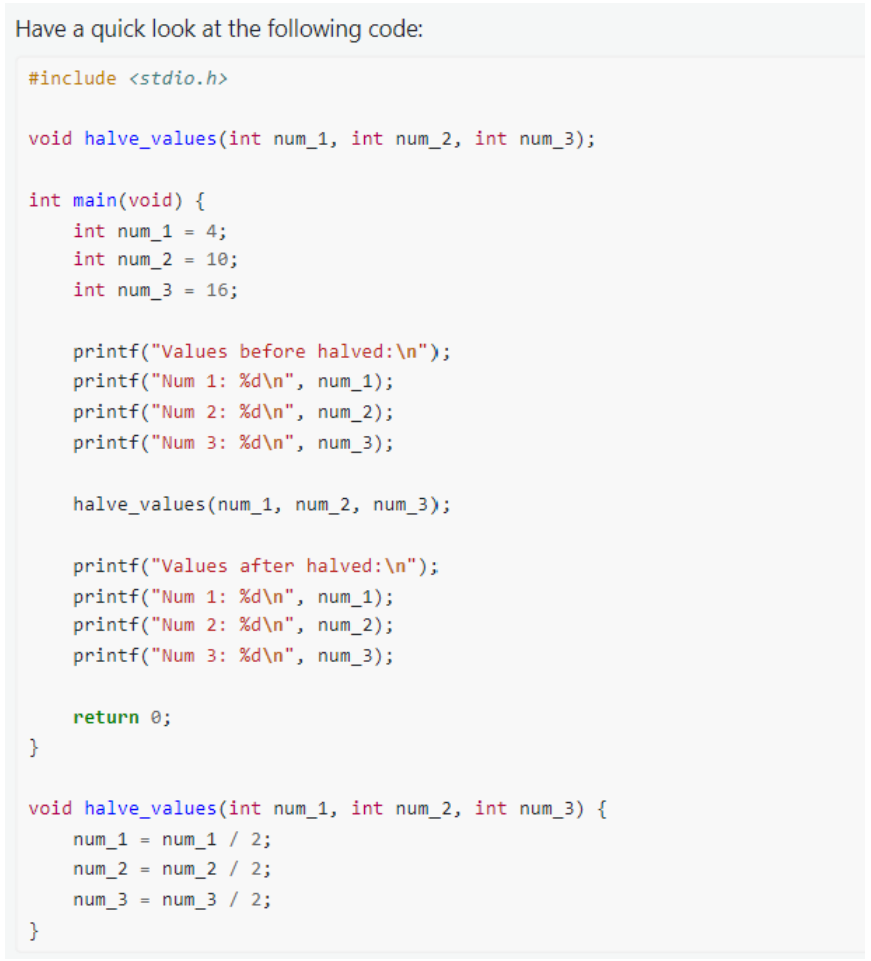
| Address | Variable |
|---|---|
| 0xFF80 | Type: ??? Name: ??? Value: value |
| 0xFF84 | Type: int Name: n Value: value |
| 0xFF88 | Type: int * Name: p Value: value |
| 0xFF8C | Type: int * Name: q Value: value |
| 0xFF90 | Type: ??? Name: ??? Value: value |

```
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.

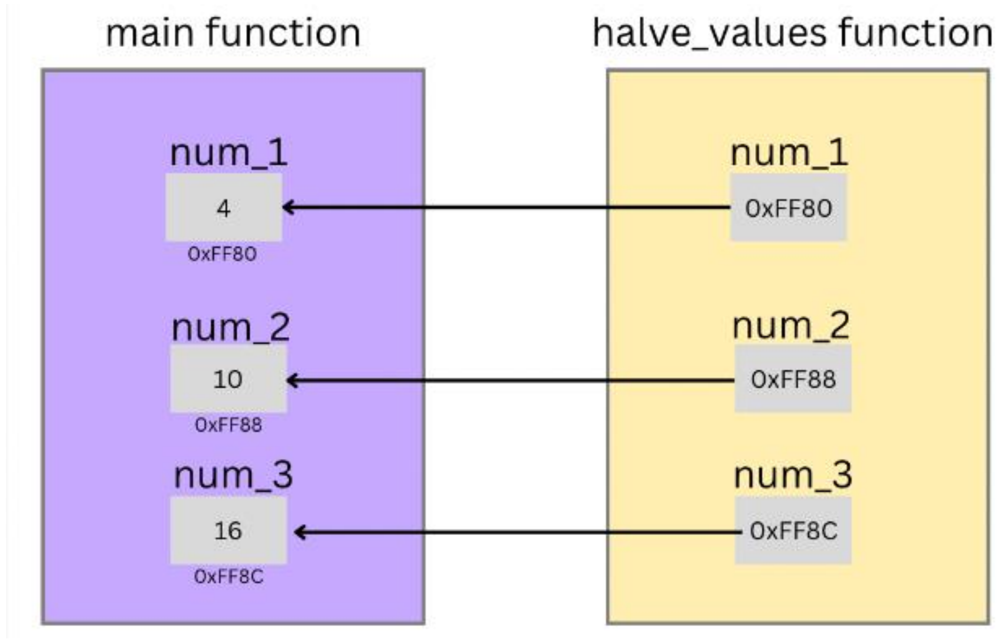
https://cgi.cse.unsw.edu.au/~cs1511/24T2/tut/07/questions

## using pointers in functions

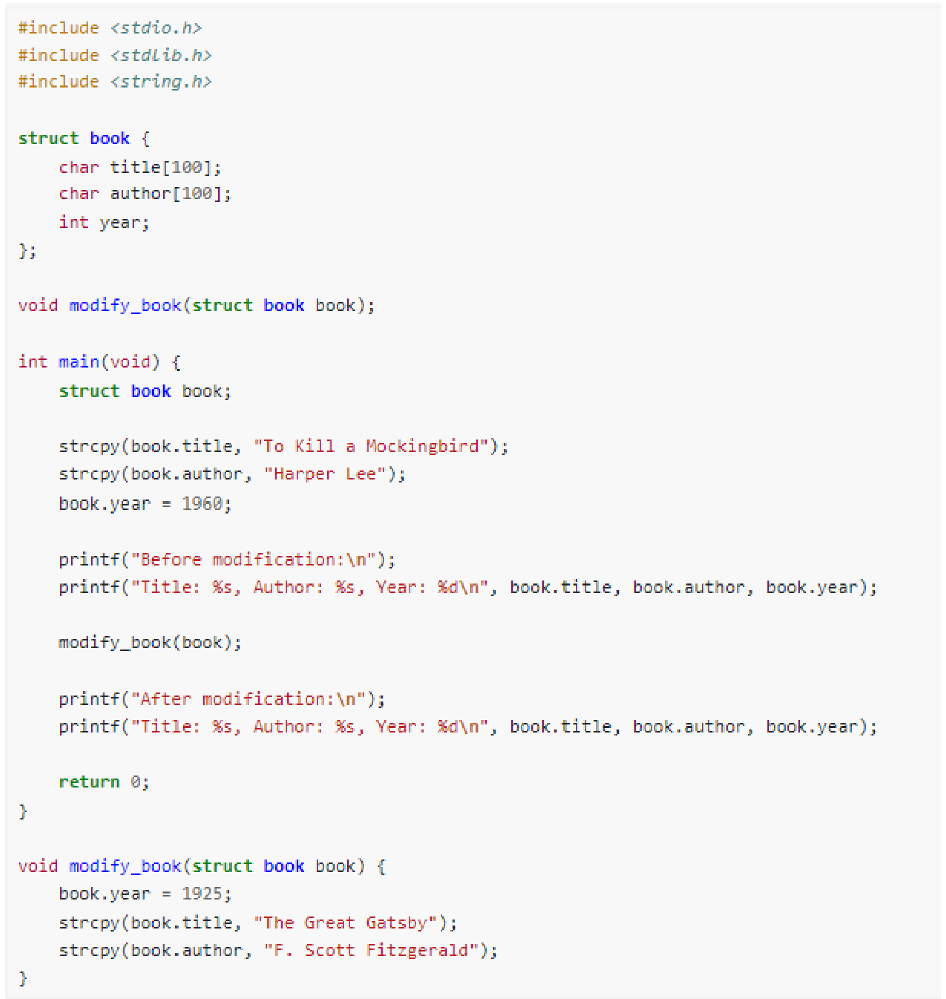

Have a quick look at the following code:

```c
#include <stdio.h>

void halve_values(int num_1, int num_2, int num_3);

int main(void) {
    int num_1 = 4;
    int num_2 = 10;
    int num_3 = 16;

    printf("Values before halved:\n");
    printf("Num 1: %d\n", num_1);
    printf("Num 2: %d\n", num_2);
    printf("Num 3: %d\n", num_3);

    halve_values(num_1, num_2, num_3);

    printf("Values after halved:\n");
    printf("Num 1: %d\n", num_1);
    printf("Num 2: %d\n", num_2);
    printf("Num 3: %d\n", num_3);

    return 0;
}

void halve_values(int num_1, int num_2, int num_3) {
    num_1 = num_1 / 2;
    num_2 = num_2 / 2;
    num_3 = num_3 / 2;
}
```

## using pointers in functions

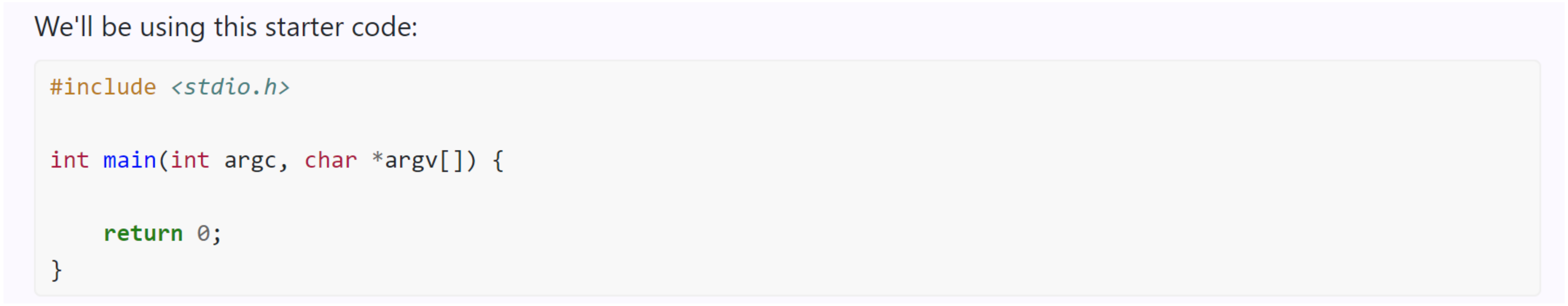


## struct pointers



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct book {
    char title[100];
    char author[100];
    int year;
};

void modify_book(struct book book);

int main(void) {
    struct book book;

    strcpy(book.title, "To Kill a Mockingbird");
    strcpy(book.author, "Harper Lee");
    book.year = 1960;

    printf("Before modification:\n");
    printf("Title: %s, Author: %s, Year: %d\n", book.title, book.author, book.year);

    modify_book(book);

    printf("After modification:\n");
    printf("Title: %s, Author: %s, Year: %d\n", book.title, book.author, book.year);

    return 0;
}

void modify_book(struct book book) {
    book.year = 1925;
    strcpy(book.title, "The Great Gatsby");
    strcpy(book.author, "F. Scott Fitzgerald");
}
```

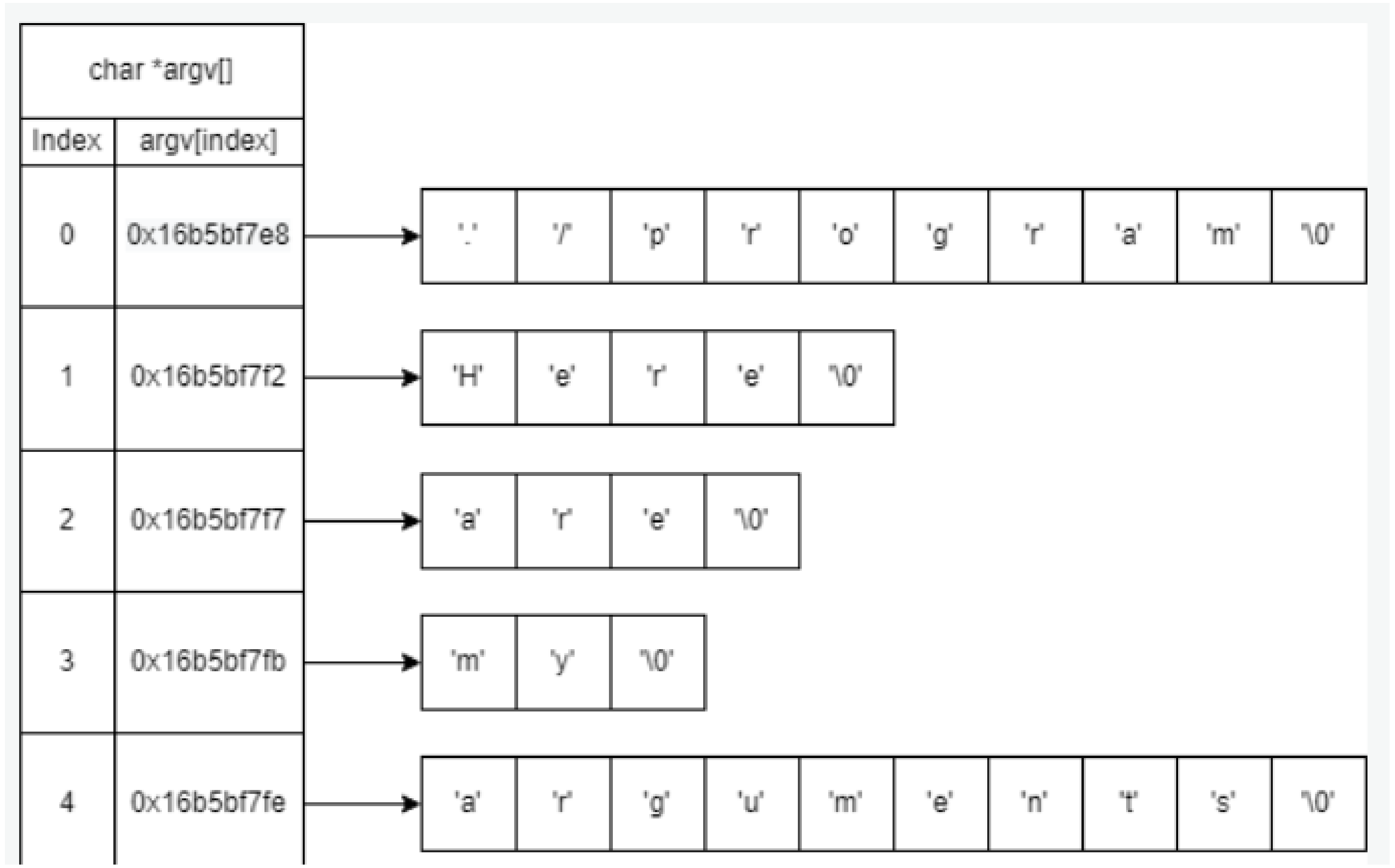
## command line arguments



We'll be using this starter code:

```c
#include <stdio.h>

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

    return 0;
}
```

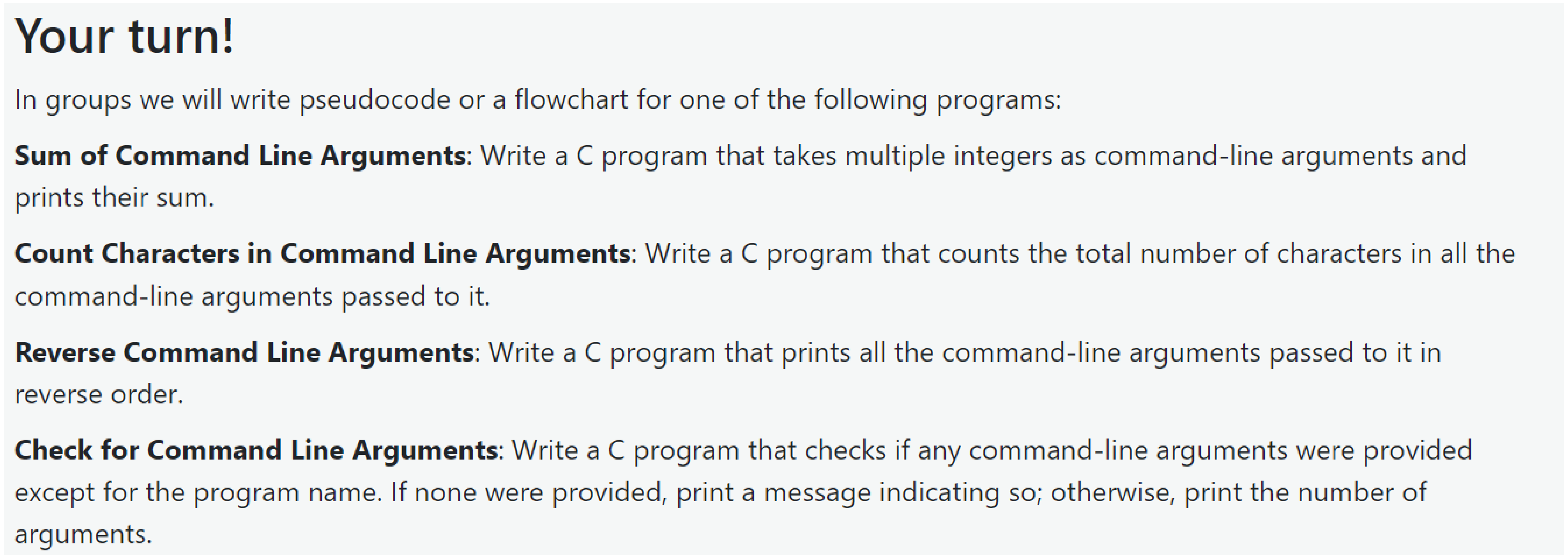
## command line arguments



## command line arguments

- how could we print out all the command line arguments?

# command line arguments



## Your turn!

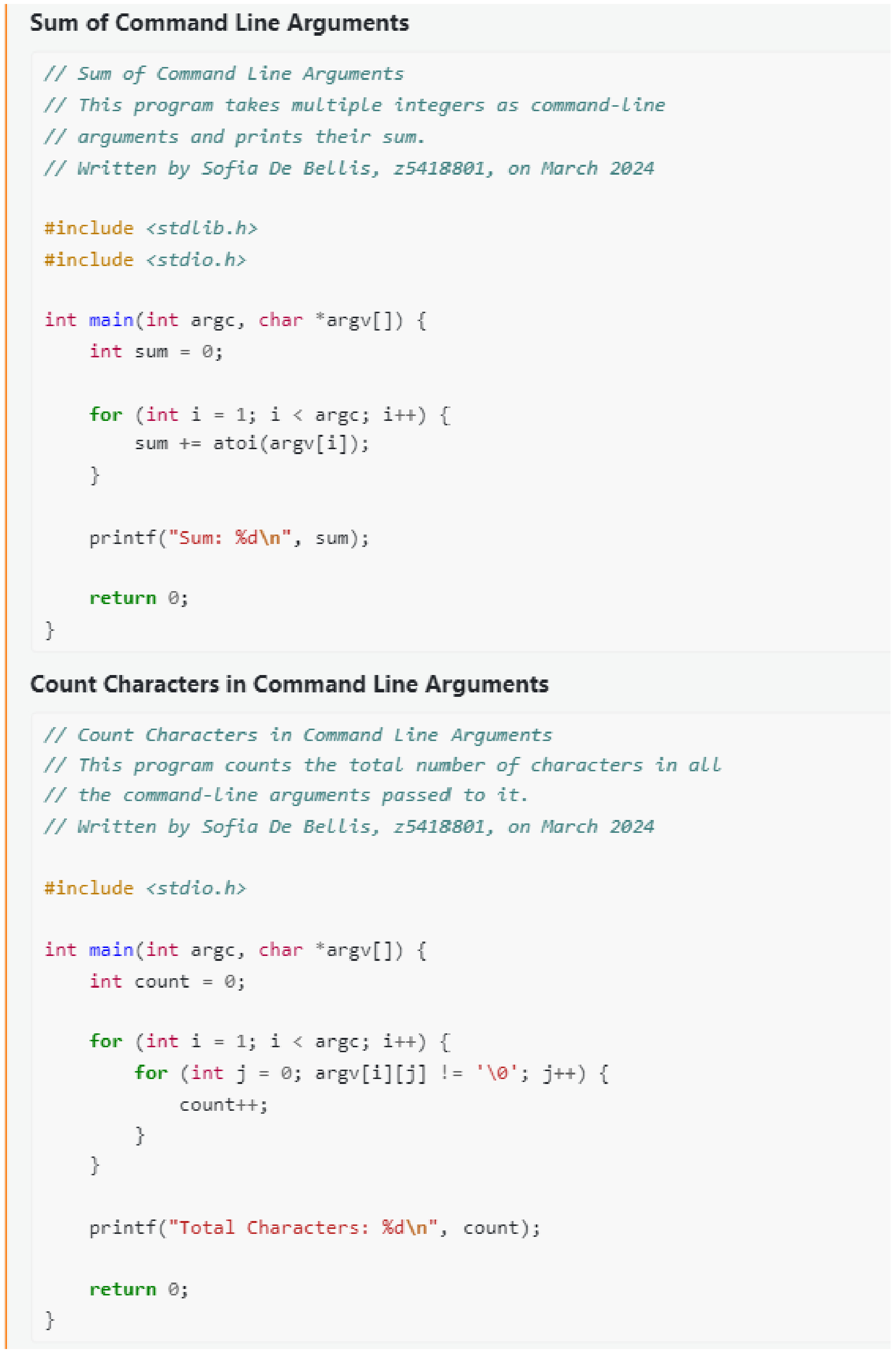
In groups we will write pseudocode or a flowchart for one of the following programs:

**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.

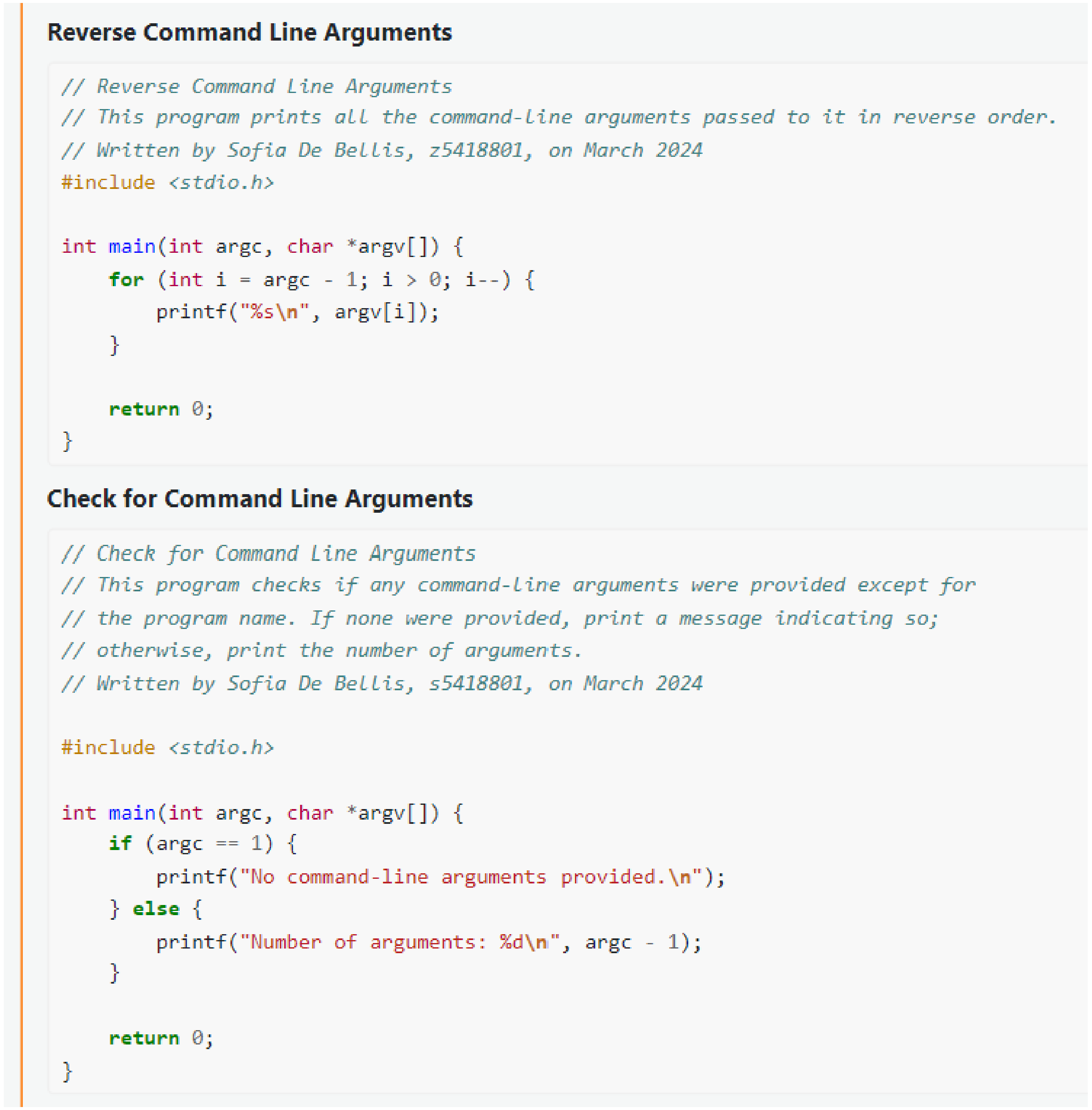
**Check for Command Line Arguments**: Write a C program that checks if any command-line arguments were provided except for the program name. If none were provided, print a message indicating so; otherwise, print the number of arguments.


### Sum of Command Line Arguments

```c
// Sum of Command Line Arguments
// This program takes multiple integers as command-line
// arguments and prints their sum.
// Written by Sofia De Bellis, z5418801, on March 2024

#include <stdlib.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    int sum = 0;

    for (int i = 1; i < argc; i++) {
        sum += atoi(argv[i]);
    }

    printf("Sum: %d\n", sum);

    return 0;
}
```

### Count Characters in Command Line Arguments

```c
// Count Characters in Command Line Arguments
// This program counts the total number of characters in all
// the command-line arguments passed to it.
// Written by Sofia De Bellis, z5418801, on March 2024

#include <stdio.h>

int main(int argc, char *argv[]) {
    int count = 0;

    for (int i = 1; i < argc; i++) {
        for (int j = 0; argv[i][j] != '\0'; j++) {
            count++;
        }
    }

    printf("Total Characters: %d\n", count);

    return 0;
}
```


### Reverse Command Line Arguments

```c
// Reverse Command Line Arguments
// This program prints all the command-line arguments passed to it in reverse order.
// Written by Sofia De Bellis, z5418801, on March 2024
#include <stdio.h>

int main(int argc, char *argv[]) {
    for (int i = argc - 1; i > 0; i--) {
        printf("%s\n", argv[i]);
    }

    return 0;
}
```

### Check for Command Line Arguments

```c
// Check for Command Line Arguments
// This program 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.
// Written by Sofia De Bellis, s5418801, on March 2024

#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc == 1) {
        printf("No command-line arguments provided.\n");
    } else {
        printf("Number of arguments: %d\n", argc - 1);
    }

    return 0;
}
```

any questions?