

COMP1511 Programming Fundamentals

Week 3 Lecture 2

Arrays

Last Lecture

- Structs with enums
- Functions - what/how/why?

Today's Lecture

- Quick Function Recap
- Style
- Handy Shorthand
- Arrays
- Look at some functions with arrays!

Link to Week 3 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/25T1/code/week_3/

Functions Recap

Functions Recap : What and Why?

What?

- A function is a block of code that performs a specific task

Why?

- Improve readability of the code
- Improve reusability of the code
- Debugging is easier (you can narrow down which function is causing issues)
- Reduces size of code (you can reuse the functions as needed, wherever needed)

Global Variables

- Variables declared outside a function have global scope
 - Do NOT use these!

```
// result is a global variable BAD DO NOT USE IN COMP1511
int result;
int main(void) {
    // answer is a local variable GOOD
    int answer;
    return 0;
}
```

Passing by Value

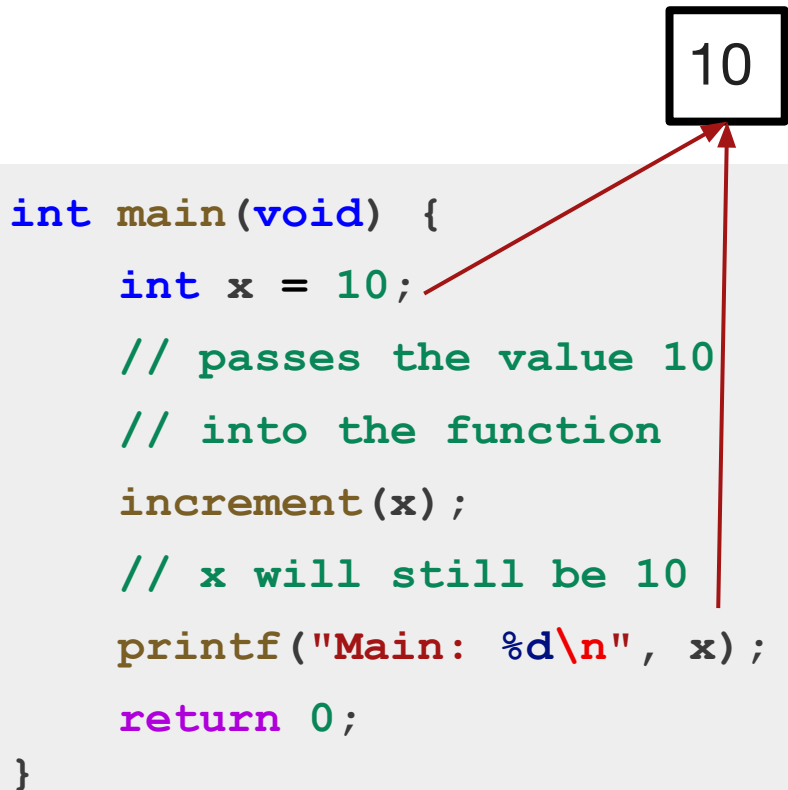
- Primitive types such as int, char, double and also enum and structs are passed by value
 - A copy of the value of the variable is passed into the function

E.g. This increment function is just modifying its own copy of x

```
void increment(int x) {  
    // modifies the  
    // local copy of x  
    x = x + 1;  
}
```

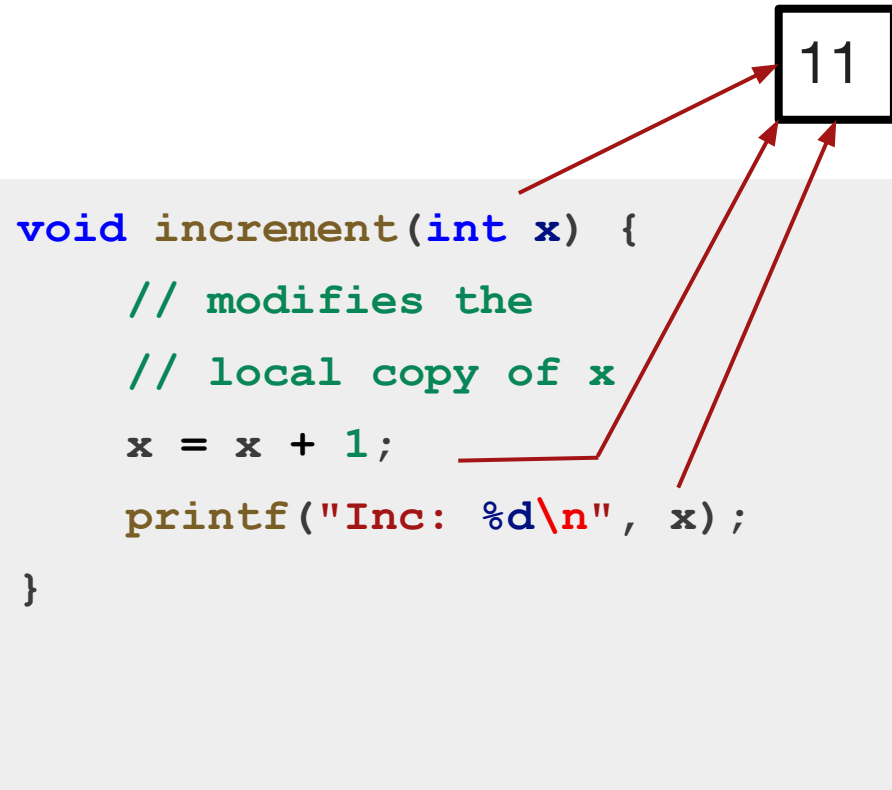

Passing by Value

```
int main(void) {  
    int x = 10;  
    // passes the value 10  
    // into the function  
    increment(x);  
    // x will still be 10  
    printf("Main: %d\n", x);  
    return 0;  
}
```



The diagram illustrates the state of memory for the 'main' function. A box containing the number '10' is positioned above the code. Two red arrows originate from the code: one points from the assignment 'int x = 10;' to the box, and the other points from the function call 'increment(x);' to the same box. This indicates that the value 10 is passed to the function as a copy, and the original value in 'main' remains unchanged.

```
void increment(int x) {  
    // modifies the  
    // local copy of x  
    x = x + 1;  
    printf("Inc: %d\n", x);  
}
```



The diagram illustrates the state of memory for the 'increment' function. A box containing the number '11' is positioned above the code. Three red arrows originate from the code: one points from the parameter 'int x' to the box, another points from the assignment 'x = x + 1;' to the box, and a third points from the 'printf' statement to the box. This shows that the function operates on a local copy of the value, which is incremented from 10 to 11, while the original value in 'main' remains 10.

Using Functions in Conditions

You can call functions inside your if statements or your while loops like this:

```
if (area_triangle(b, h) < 10) {  
    ...  
}
```

```
while (scanf("%d", &n) == 1) {  
    ...  
}
```

Note: You can't do this with functions that have void return types

Some handy shorthand!!

Increment and Decrement

```
// Increment count by 1  
count = count + 1;  
count++;
```

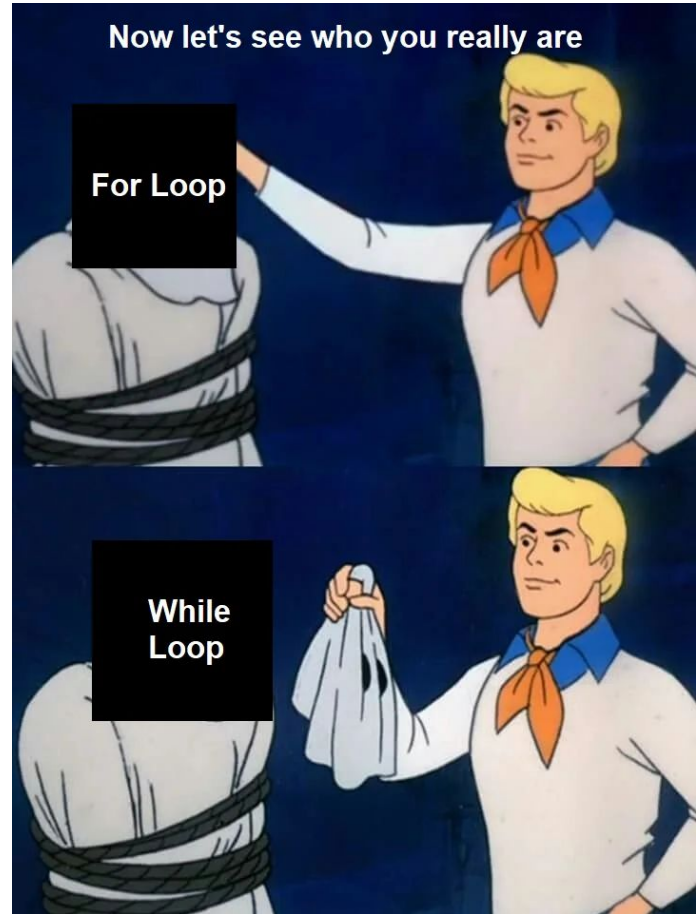
```
// Decrement count by 1  
count = count - 1;  
count--;
```

```
// Increment count by 5  
count = count + 5;  
count += 5;
```

```
// Decrement count by 5  
count = count - 5;  
count -= 5;
```

for loops

- Very similar to **while** loops!
- You can do everything you need with a while loop
- **for** loops are really just a short hand for while loops in C
- **for** loops are very handy for loops when you know the number of iterations you need!
 - counting loops



For loop structure

initialisation:

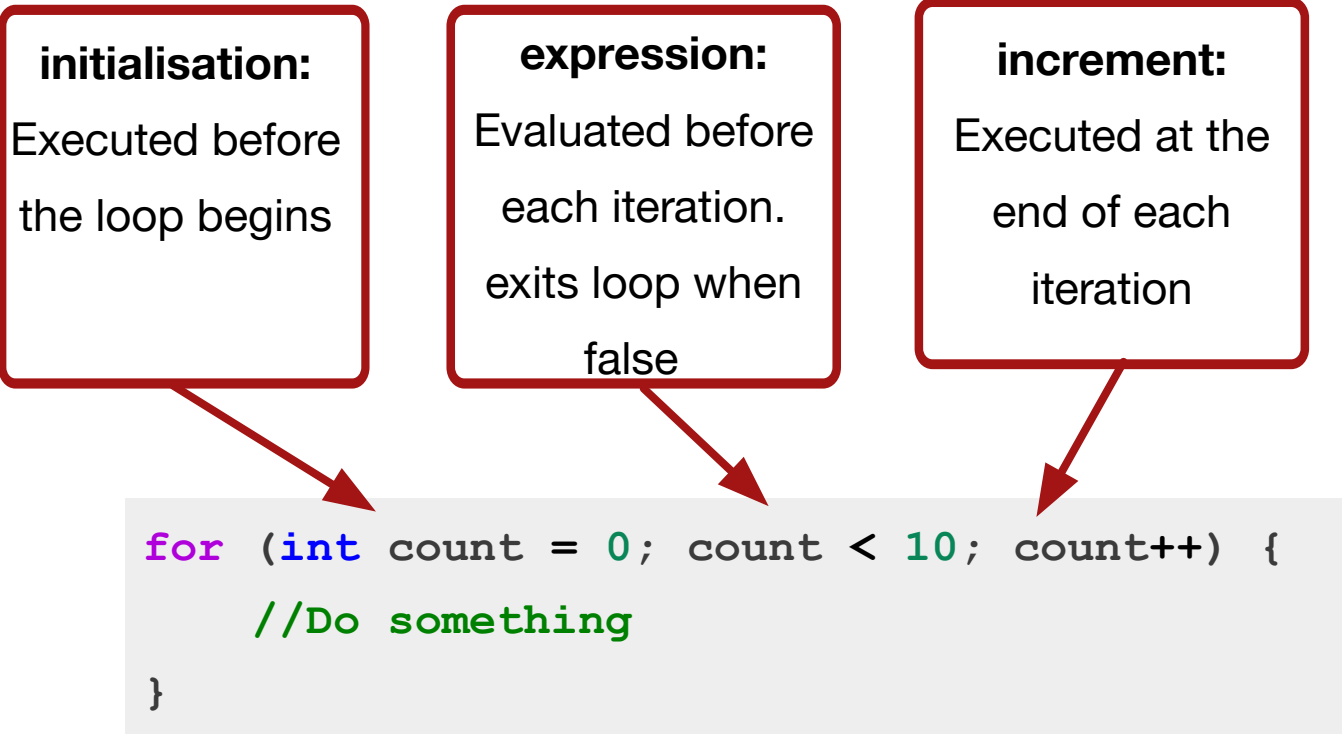
Executed before
the loop begins

expression:

Evaluated before
each iteration.
exits loop when
false

increment:

Executed at the
end of each
iteration



```
for (int count = 0; count < 10; count++) {  
    //Do something  
}
```

while loop vs for loop

These two loops do exactly the same thing!

```
int i = 0;
while (i < 10) {
    printf("%d\n", i);
    i++;
}
```

```
for (int i = 0; i < 10; i++) {
    printf("%d\n", i);
}
```

What is Style? Why Style?

- The code we write is for human eyes
- We want to make our code:
 - easier to read
 - easier to understand
- Coding should always be done in style - it is worth it...
 - ensures less possibility for mistakes
 - ensures faster development time
 - You also get marks for style in assignments
 - If we need to mark your code in the final manually it is good if it is not a dog's breakfast

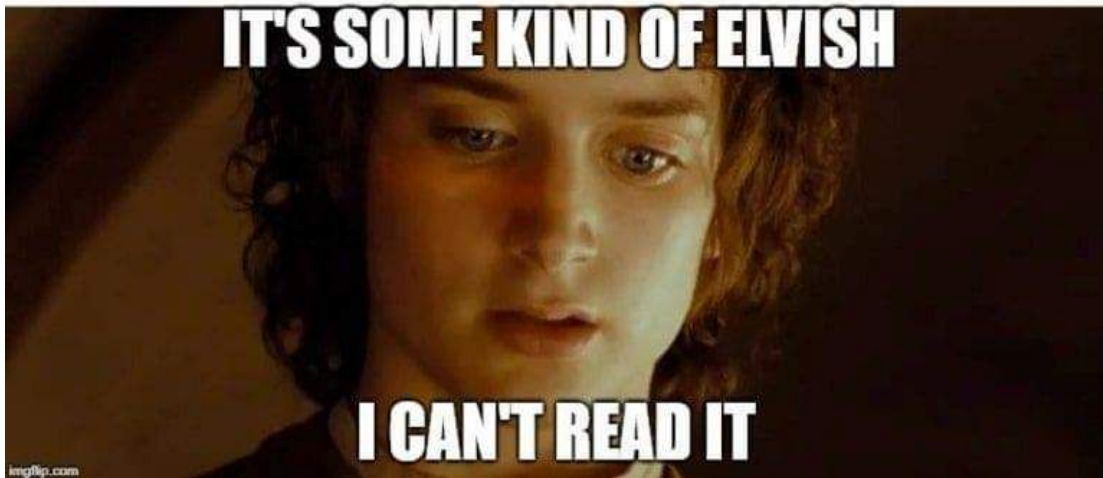
What is Good Style?



- Indentation and Bracketing
- Names of variables and functions
- Structuring your code
- Nesting
- Repetition
- Comments
- Consistency

Bad Style Demo

When you trying to look at
the code you wrote a month ago



Let's look at
`bad_style.c`

- What are some things we should fix?

Clean as you go

- Write comments where they are needed
- Name your variables based on what that variable is there to do
- In your block of code surrounded by {}:
 - Indent 4 spaces
 - Vertically align closing bracket with statement that opened it
- One expression per line
- Consistency in spacing
- Watch your code width (≤ 80 characters)
- Watch the nesting of IFs - can it be done more efficiently?
- Break code into functions

Style Guide

- Often different organisations you work for, will have their own style guides, however, the basics remain the same across
- Your assignment will have style marks attached to it
- We have a style guide in 1511 that we encourage you to use to establish good coding practices early:
- https://cgi.cse.unsw.edu.au/~cs1511/25T1/resources/style_guide.html

Arrays

**What if you wanted to store many
related values of the same type?**

Number of Chocolates Eaten

```
int day_1 = 2;
int day_2 = 3;
int day_3 = 3;
int day_4 = 5;
int day_5 = 7;
int day_6 = 1;
int day_7 = 3;
// Any day with 3 or more is too much!
if (day_1 >= 3) {
    printf("Too many chocolates\n");
}
if (day_2 >= 3) {...
```

Does this seem repetitive? What if I tracked a year's worth??!

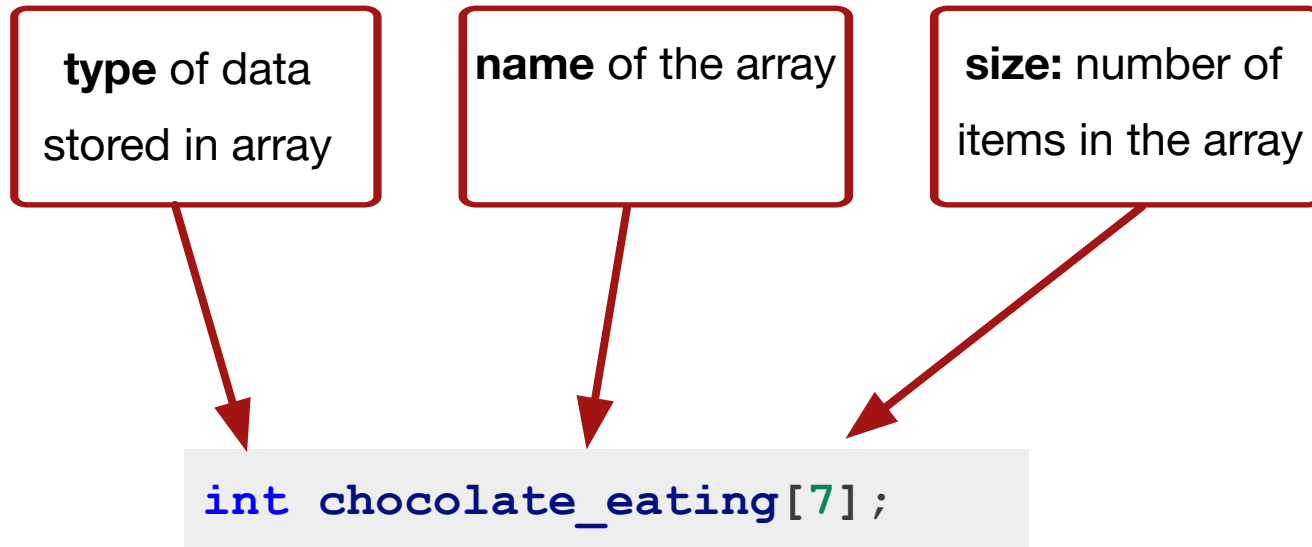
Data Structures

- A data structure is a way of organizing and storing data so that it can be accessed and used efficiently
- In this course we will learn about two pretty cool data structures:
 - Arrays (NOW!)
 - Linked Lists (after flexibility week)
- There are other data structures that you will learn about in further computing courses
- Choosing the right data structure depends on what the problem is and what you are trying to achieve.

Arrays!

- A collection of variables all of the same type (homogenous)
 - Think about how this is very different to a struct
- A contiguous data structure
 - All data in an array is stored in consecutive memory locations
- A random access data structure
 - We can access any data in the collection directly without having to scan through other data elements
- An indexed structure
 - We just have one variable identifier for the whole collection of data
 - We can use indexes to access specific pieces of data

Declaring an Array



- This declares an array named chocolate_eating, that can store 7 integers

Declaring and Initialising an Array

```
// This declares an array named chocolate_eating,  
// that can store 7 integers and initialises  
// their values to 4, 2, 5, 2 and so on.  
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};
```

```
// This would declare the array and  
// initialise all values to 0  
int chocolate_eating[7] = {};
```

Declaring and Initialising an Array

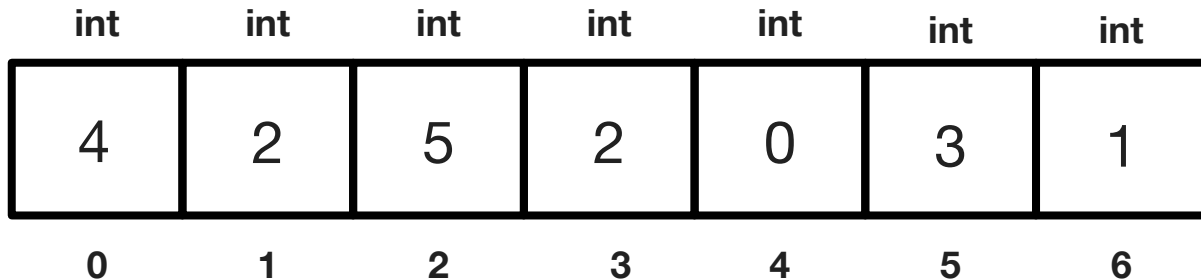
```
// This is illegal and does not compile
// You can only use this initialisation syntax
// when you declare the array
// NOT later
int chocolate_eating[7];
chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};
// This is the correct way all in one line
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};
```

Visualising an Array

So let's say we have this declared and initialised:

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};
```

This is what it looks like visually:

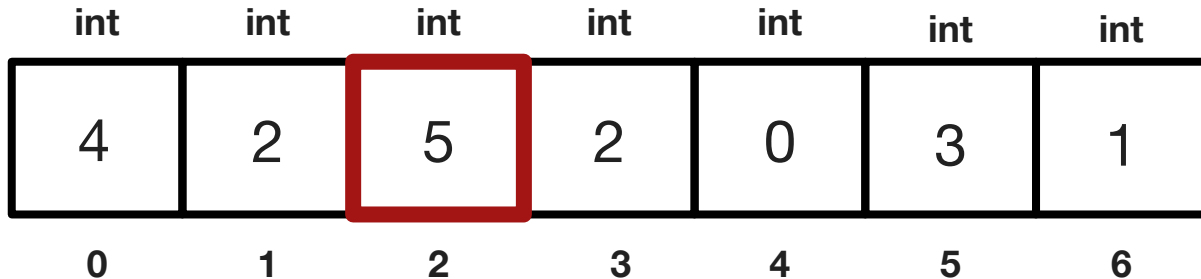


Note: The array holds 7 elements. Indexes start at 0

Accessing Elements in an Array

- You can access any element of the array by using its index
 - Indexes start from 0
 - Trying to access an index that does not exist, will result in an error

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};
```



`chocolate_eating[2]` would access the third element

Accessing Elements in an Array

- You can access any element of the array by using its index
 - Indexes start from 0
 - Trying to access an index that does not exist, will result in an error

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};
```

int	int	int	int	int	int	int
4	2	5	2	0	3	1
0	1	2	3	4	5	6

`chocolate_eating[7]` would cause a run-time error

A closer look at arrays

- You can't printf() a whole array
 - but you can print individual elements
- You can't scanf() a whole array at once
 - but you can scanf() individual elements
- You can't assign a whole array to another array variable
 - but you can create an array and copy the individual elements

```
int a[7] = {4, 2, 5, 2, 0, 3, 1};  
int b[7] = a;    // You can't do this!
```


Printing elements in an array

Does this look repetitive?

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
printf("%d ", chocolate_eating[0]);  
printf("%d ", chocolate_eating[1]);  
printf("%d ", chocolate_eating[2]);  
printf("%d ", chocolate_eating[3]);  
printf("%d ", chocolate_eating[4]);  
printf("%d ", chocolate_eating[5]);  
printf("%d ", chocolate_eating[6]);
```

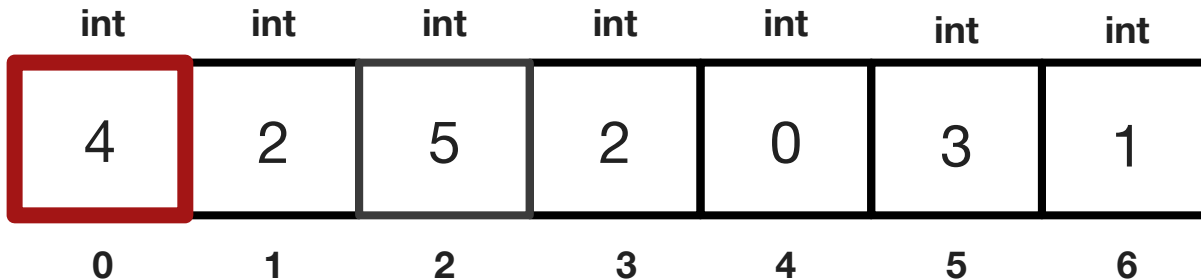
How could we do this in a better way?

Traversing an Array

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
int i = 0;  
while (i < 7) {  
    printf("%d ", chocolate_eating[i]);  
    i++;  
}
```

Start at index 0

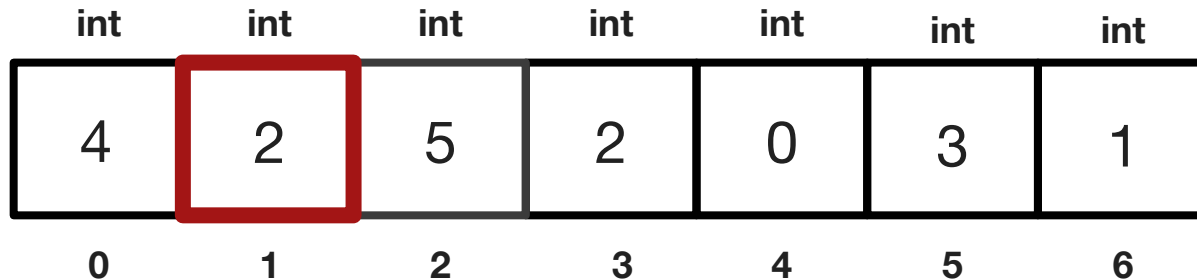
chocolate_eating[0]



Traversing an Array

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
int i = 0;  
while (i < 7) {  
    printf("%d ", chocolate_eating[i]);  
    i++;  
}
```

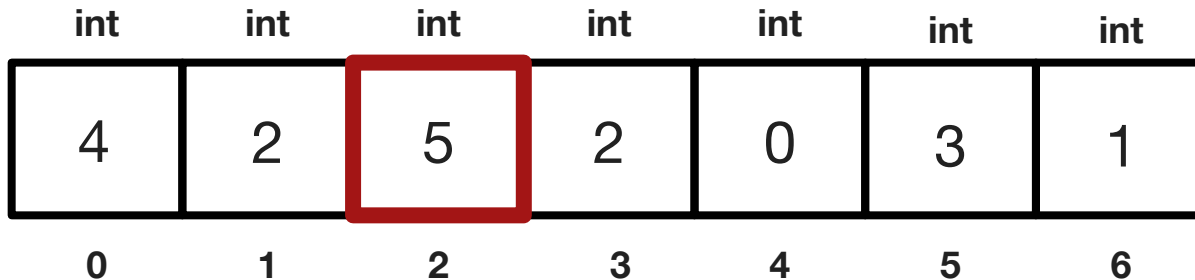
Increment index by 1
chocolate_eating[1]



Traversing an Array

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
int i = 0;  
while (i < 7) {  
    printf("%d ", chocolate_eating[i]);  
    i++;  
}
```

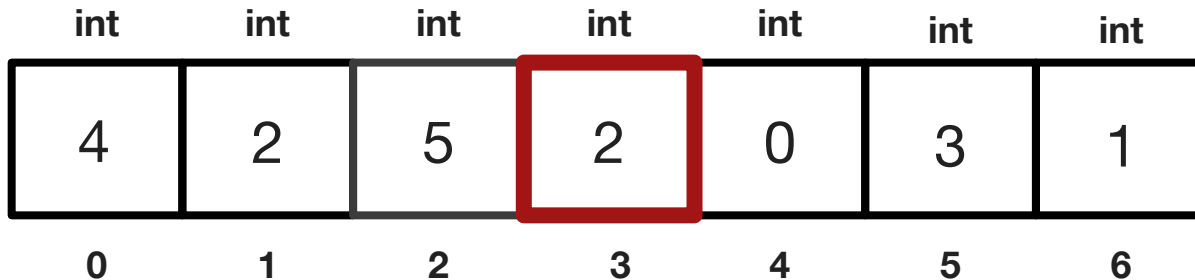
Increment index by 1
chocolate_eating[2]



Traversing an Array

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
int i = 0;  
while (i < 7) {  
    printf("%d ", chocolate_eating[i]);  
    i++;  
}
```

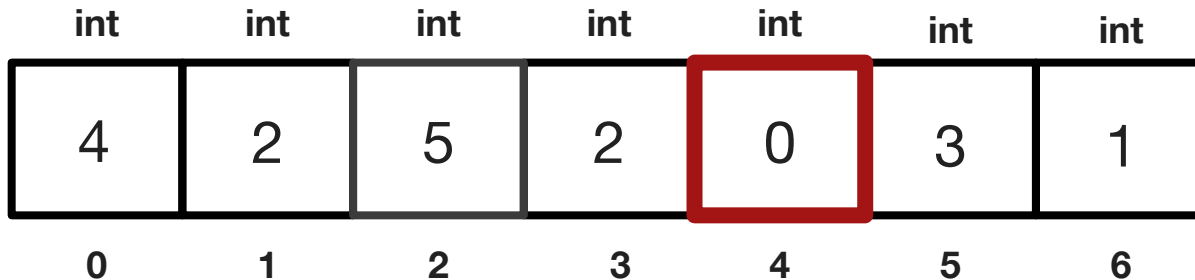
Increment index by 1
chocolate_eating[3]



Traversing an Array

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
int i = 0;  
while (i < 7) {  
    printf("%d ", chocolate_eating[i]);  
    i++;  
}
```

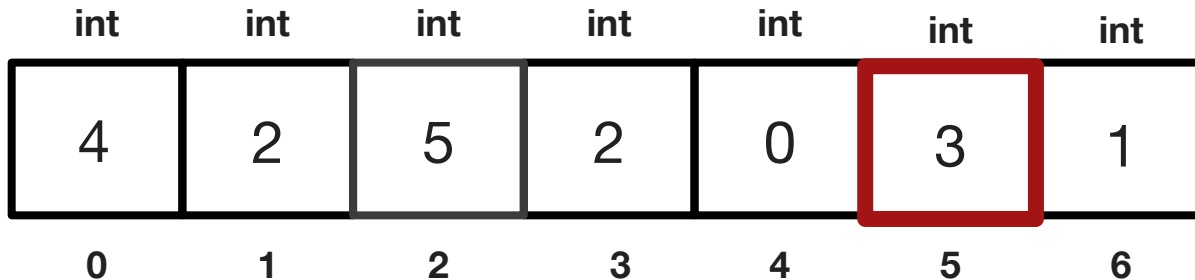
Increment index by 1
chocolate_eating[4]



Traversing an Array

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
int i = 0;  
while (i < 7) {  
    printf("%d ", chocolate_eating[i]);  
    i++;  
}
```

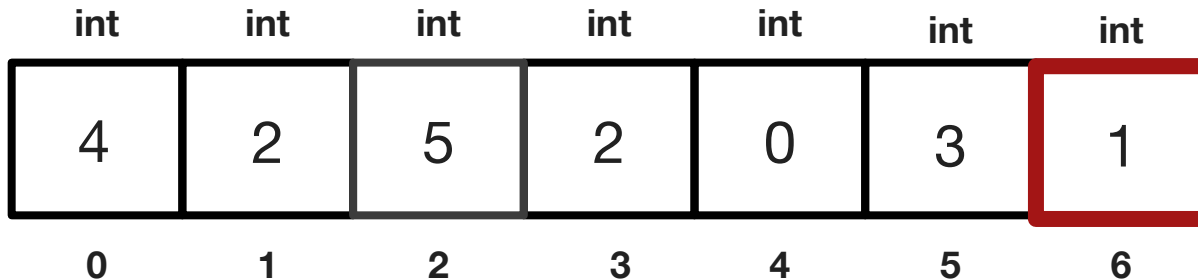
Increment index by 1
chocolate_eating[5]



Traversing an Array

```
int chocolate_eating[7] = {4, 2, 5, 2, 0, 3, 1};  
int i = 0;  
while (i < 7) {  
    printf("%d ", chocolate_eating[i]);  
    i++;  
}
```

Increment index by 1
chocolate_eating[6]



Demo arrays!

`simple_array.c`

`numbers.c`

- print array, (while loop and for loop)

- sum,

- average,

- divisible by 4,

- multiply by 2,

- scan in numbers

`numbers_functions.c`

Arrays and Functions

- We can pass arrays into functions!
- The function needs a way of knowing the size of the array

```
// Can pass in array of int of any size  
void print_array(int size, int array[]);
```

Arrays and Functions

```
void print_array(int size, int array[]);  
  
int main(void) {  
    int marks[] = {9, 8, 10, 2, 7};  
    int ages[] = {21, 42, 11};  
    print_array(5, marks);  
    print_array(3, ages);  
    return 0;  
}  
  
void print_array(int size, int array[]) {  
    for (int i = 0; i < size; i++) {  
        printf("%d ", array[i]);  
    }  
}
```

Arrays and Functions

- Functions do not get a copy of all the array values passed into them.
- They can access the original array from the calling function
- This means they can modify the values directly from the function
- More about this in future weeks!

Arrays and Functions

- We can pass an array into a function and initialise all the values like this!!

```
int main(void) {  
    int marks[SIZE];  
    scan_marks(SIZE, marks);  
    print_marks(SIZE, marks);  
    return 0;  
}  
  
void scan_marks(int size, int array[]) {  
    for (int i = 0; i < size; i++) {  
        scanf("%d ", &array[i]);  
    }  
}
```

Arrays and Functions

- Trying to return an array from a function by doing something like this looks ok but **fails** spectacularly!
- We will explain this in more detail later in the course

```
// You can't return an array like
// this from a function
int[] scan_marks(void) {
    int array[SIZE];
    for (int i = 0; i < SIZE; i++) {
        scanf("%d ", &array[i]);
    }
    return array;
}
```

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.



<https://forms.office.com/r/PcEMQSXP61>

What did we learn today?

- Functions recap (pass_by_value.c scanf_loop.c)
- Arrays (simple_array.c numbers.c)
- Arrays with Functions (numbers_functions.c)

Next Week

- Lectures:
 - 2D arrays
 - strings
- Assignment 1 will be released next week
 - Material covered in lectures next week will be very important

Reach Out

Content Related Questions:
Forum

Admin related Questions email:
cs1511@unsw.edu.au

