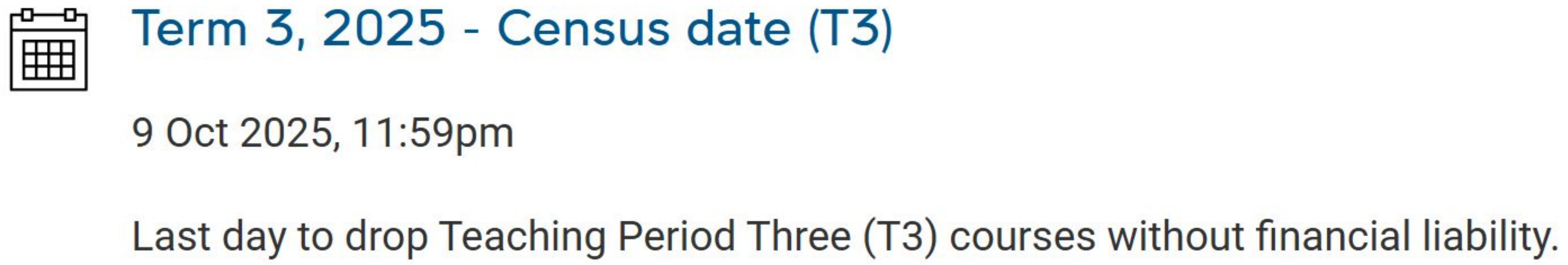
COMP1511/1911 Programming Fundamentals

Week 3 Lecture 2

Arrays

Census Date



Term 3, 2025 - Census date (T3)

9 Oct 2025, 11:59pm

Last day to drop Teaching Period Three (T3) courses without financial liability.

About Census Dates | UNSW Current Students

Public Holiday on Monday

- Booking for tut/lab
- Recording for lecture
- Lab 3 due - week 4 tuesday 6pm
- Lab 4 due - week 5 tuesday 6pm
- Assignment released early week 4

Revision Sessions Week 4

- Like a Hybrid tutorial/lab session
    - structured with lab style questions
- Forum post coming soon with more information!!

Link to Week 3 Live Lecture Code

https://cgi.cse.unsw.edu.au/~cs1511/25T3/code/week_3/

Disclaimer:

Some live lecture code is not cleaned up and polished!!!

It may have some things that are not 100% perfect style.

I also sometimes have extra comments explaining how

C works that would not be needed usually.

Yesterday’s Lecture

- Nested while loop, struct, enum recap
- Functions!

Today’s Lecture

- Function Recap
- Function, Memory and Scope
- Style
- Handy Shorthand
- Arrays (A hurdle topic)
- If we have time, look at some functions with arrays!

Functions Recap

Functions

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

Benefits of functions

- **Modularity**: Breaks complex programs into simpler, manageable pieces, easier to read and understand
- **Reusability**: Avoids code duplication, as you can reuse the functions
- **Abstraction**: Hides the implementation details and allows you to focus on higher-level logic.
- Allow us to **test** and **debug** smaller chunks of code in isolation

Recap: Simple Functions

```c
double add_numbers (double x, double y);

int main(void) {
    int x = 9;
    double answer = add_numbers(1.5, x);
    printf("The answer is %lf\n", answer);
    return 0;
}

// This function returns the sum of 2 given doubles
double add_numbers (double x, double y) {
    double sum;
    sum = x + y;
    return sum;
}
```

Memory and Scope

- Blocks of code in C are delimited by a pair if braces {}.
  - The body of a function is a common example of a block.
- Generally the scope of a variable is
  - Between where the variable is declared
  - The end of the block it was declared in
- Variables declared inside functions are called local variables.
- Code demos: memory_scope.c

Functions and Local Variables

- Local variables are created when the function called and destroyed when function returns
- A function’s variables are not accessible outside the function

```c
double add_numbers(double x, double y) {
    // sum is a local variable
    double sum;
    sum = x + y;
    return sum;
}
```

Global Variables

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

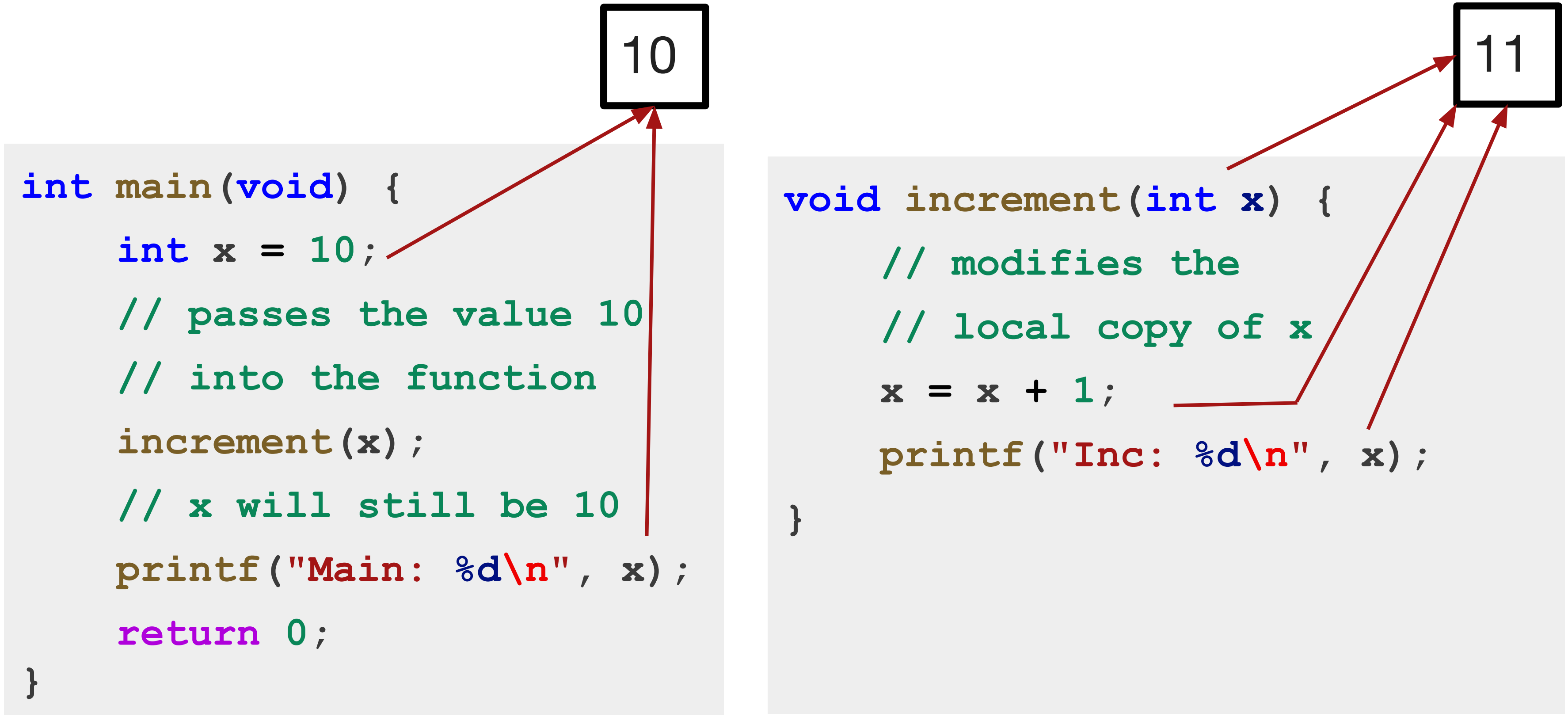
```
// result is a global variable BAD DO NOT USE IN COMP1511
int result;
int main(void) {
    // answer is a local variable GOOD
    int answer;
    return 0;
}
```

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
    - This increment function is just modifying its own copy of x
    - Code demo: pass_by_value.c

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

Passing by Value



```c
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;
}
```

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

Using Functions in Conditions

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

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

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

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

Style

- The code we write is for human eyes
- We want to make our code:
    - easier to read
    - easier to understand

Style Guide

- Often different organisations you work for, will have their own style guides, however, the basics remain the same across
- 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/~cs1511/25T3/resources/style_guide.html

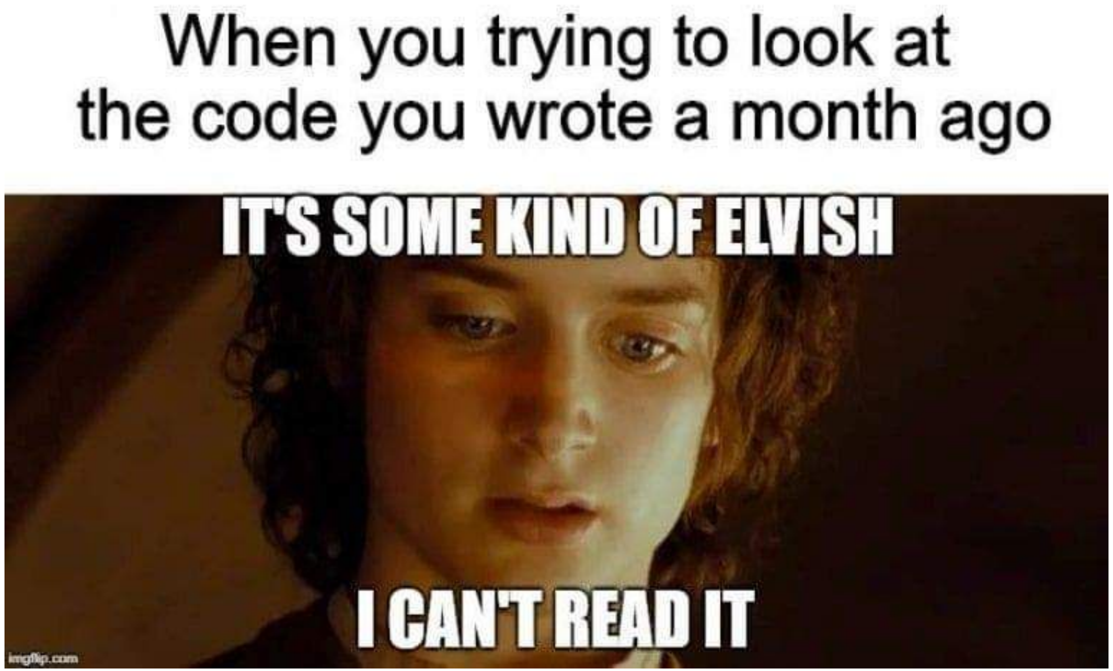
Benefits of Good Style

- less possibility for mistakes
- helps with 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



Let's look at bad_style.c

What are some things we should fix?

Tips: 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

Some handy shorthand!!

Increment and Decrement

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

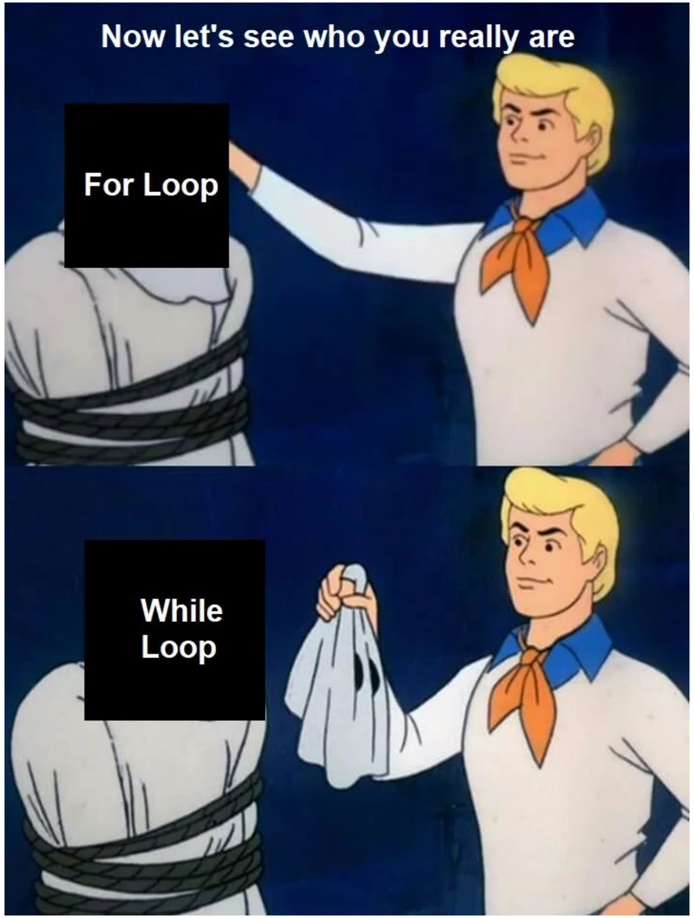
```
// Decrement count by 1
count = count - 1;
count--;
```

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

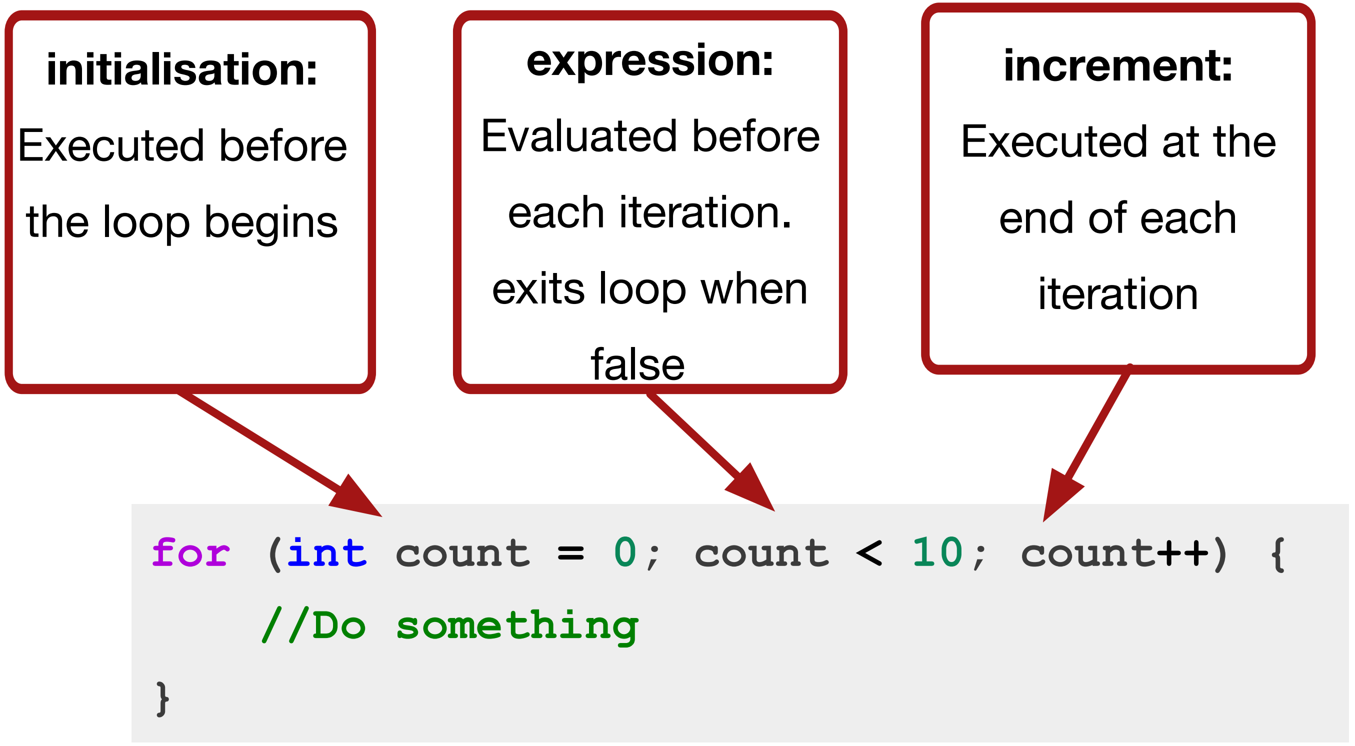
```
// Decrement count by 5
count = count - 5;
count -= 5;
```

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

```c
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);
}
```

Arrays

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

Number of Chocolates Eaten

```c
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??!!

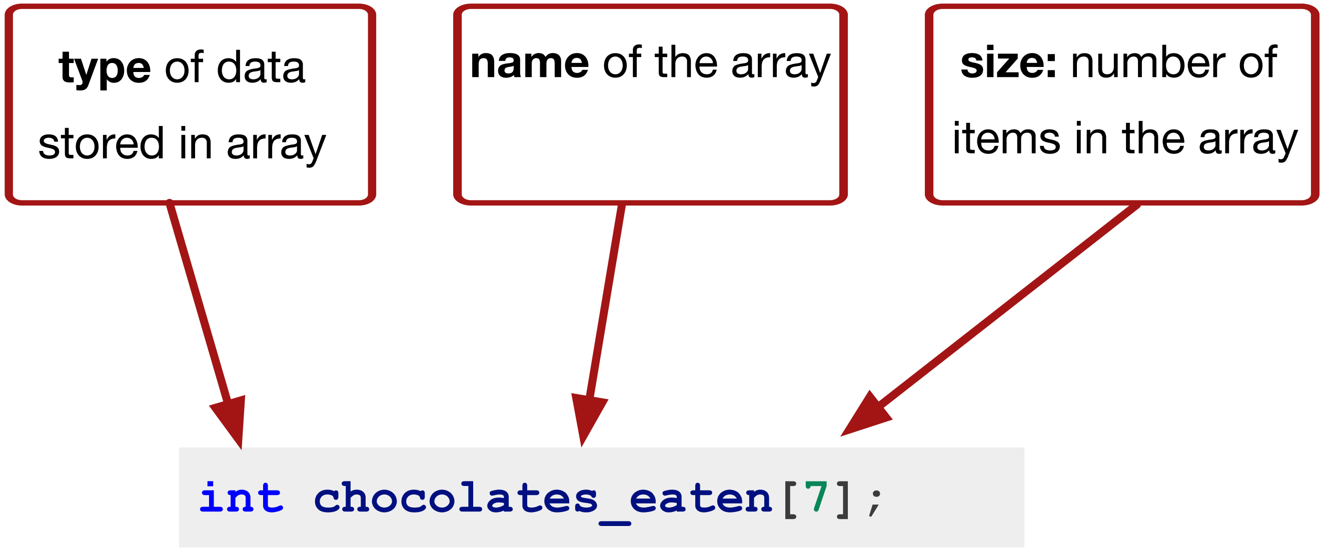
Data Structures

- 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 very important 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 uses indexes to access specific pieces of data

Declaring an Array


type of data stored in array
name of the array
size: number of items in the array

```
int chocolates_eaten[7];
```

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

Declaring and Initialising an Array

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

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

Declaring and Initialising an Array

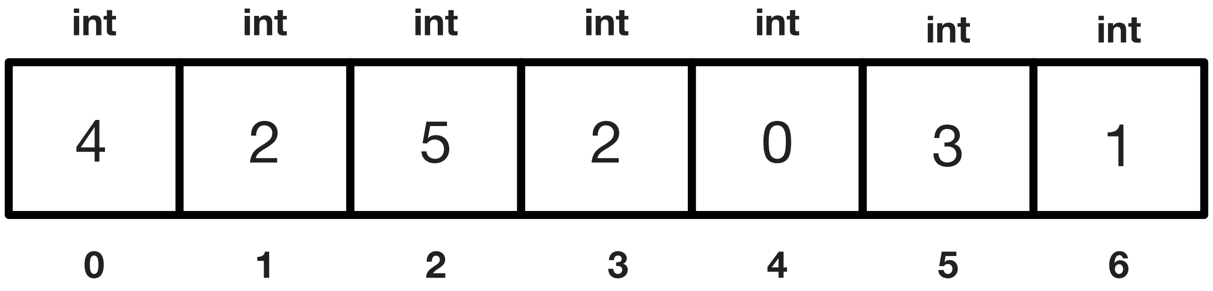
```
// This is illegal and does not compile
// You can only use this initialisation syntax
// when you declare the array
// NOT later
int chocolates_eaten[7];
chocolates_eaten[7] = {4, 2, 5, 2, 0, 3, 1};
// This is the correct way all in one line
int chocolates_eaten[7] = {4, 2, 5, 2, 0, 3, 1};
```

Visualising an Array

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

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

This is what it looks like visually:

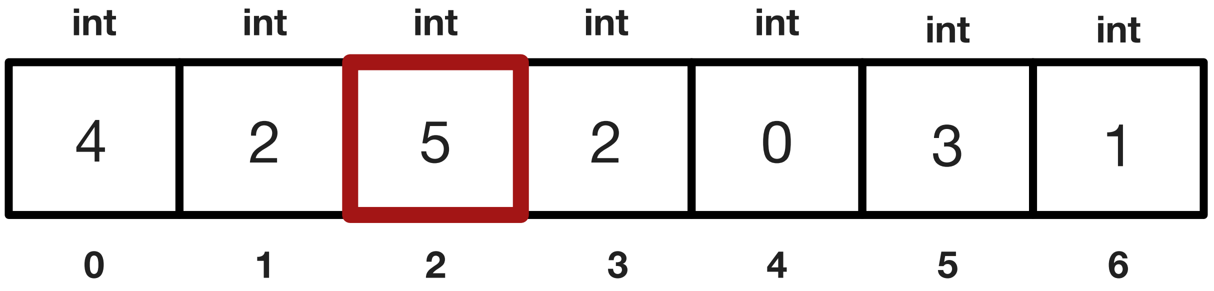


Note: The array holds 7 elements. Indexes start at 0

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

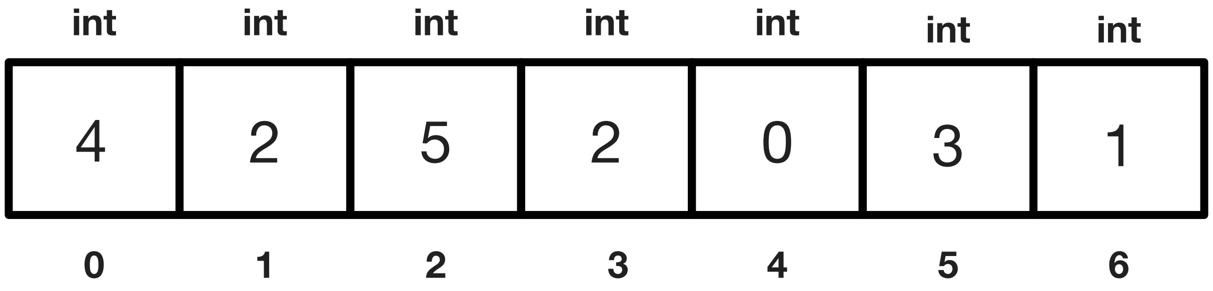


`chocolates_eaten[2]` would access the third element

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



`chocolates_eaten[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?

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

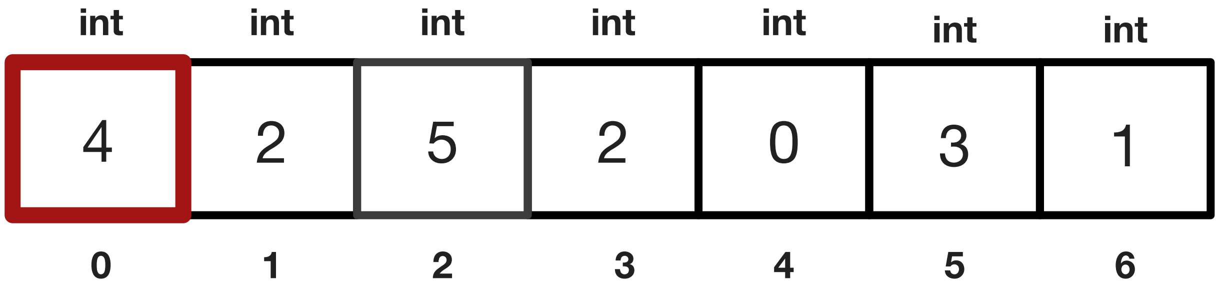
How could we do this in a better way?

Traversing an Array

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

Start at index 0

`chocolates_eaten[0]`

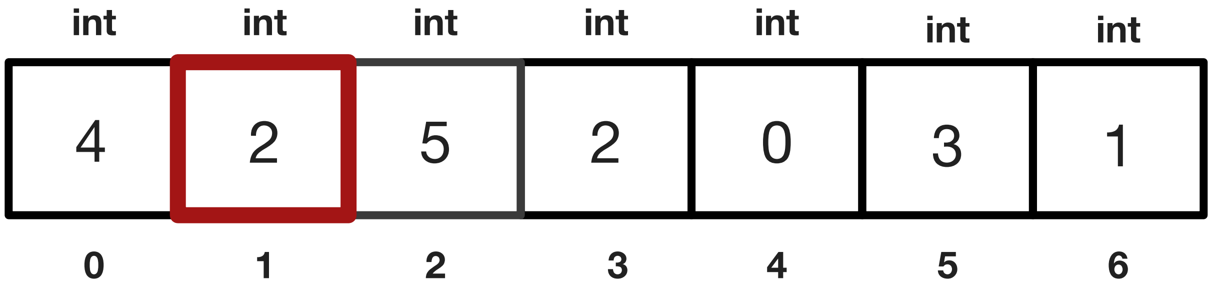


Traversing an Array

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

Increment index by 1

`chocolates_eaten[1]`

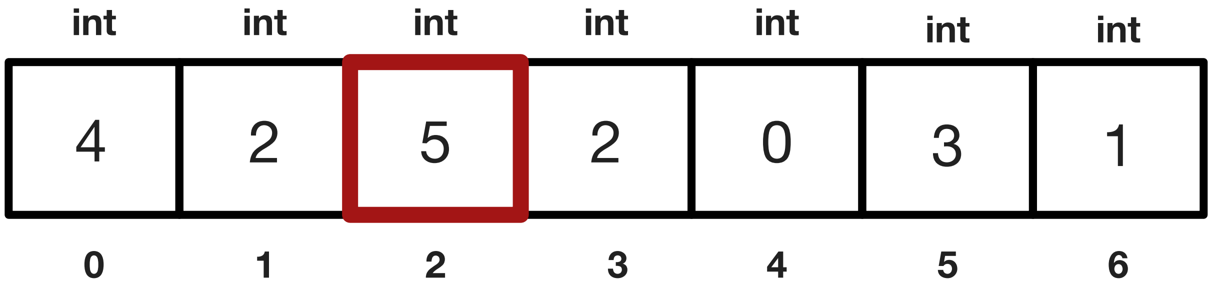


Traversing an Array

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

Increment index by 1

`chocolates_eaten[2]`

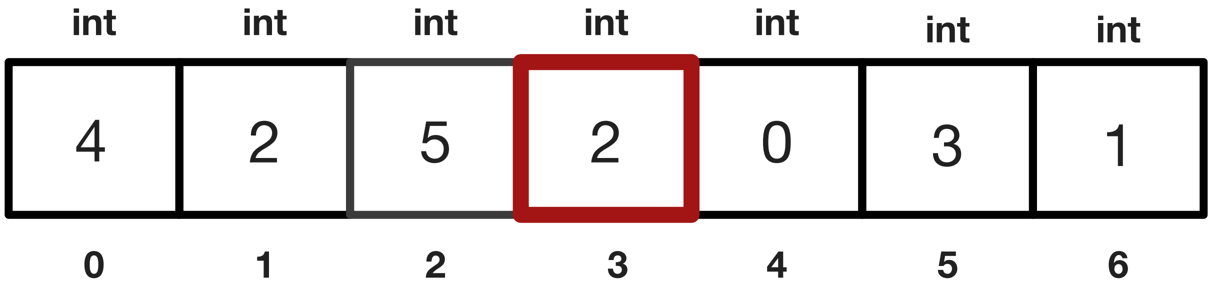


Traversing an Array

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

Increment index by 1

`chocolates_eaten[3]`

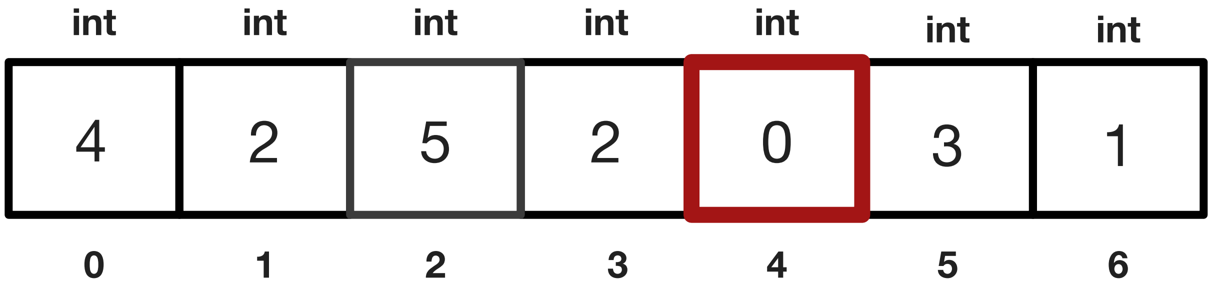


Traversing an Array

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

Increment index by 1

`chocolates_eaten[4]`

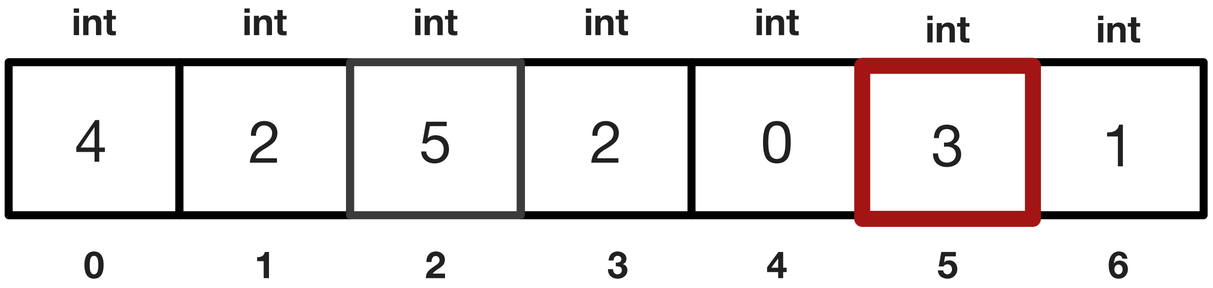


Traversing an Array

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

Increment index by 1

`chocolates_eaten[5]`

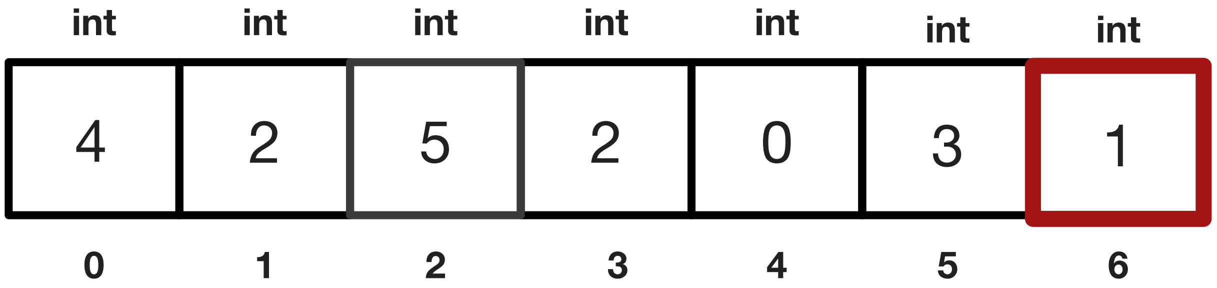


Traversing an Array

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

Increment index by 1

`chocolates_eaten[6]`



Demo arrays!

`simple_array.c`

`numbers.c`

scan in numbers

print array, (while loop and for loop)

sum,

add 10 to all values,

`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

```c
// 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;
}
```

What did we learn today?

- Functions recap (memory_scope.c 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

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

Reach Out

Content Related Questions:

Forum

Admin related Questions email:

cs1511@unsw.edu.au