

DPST1091 / CPTG1391

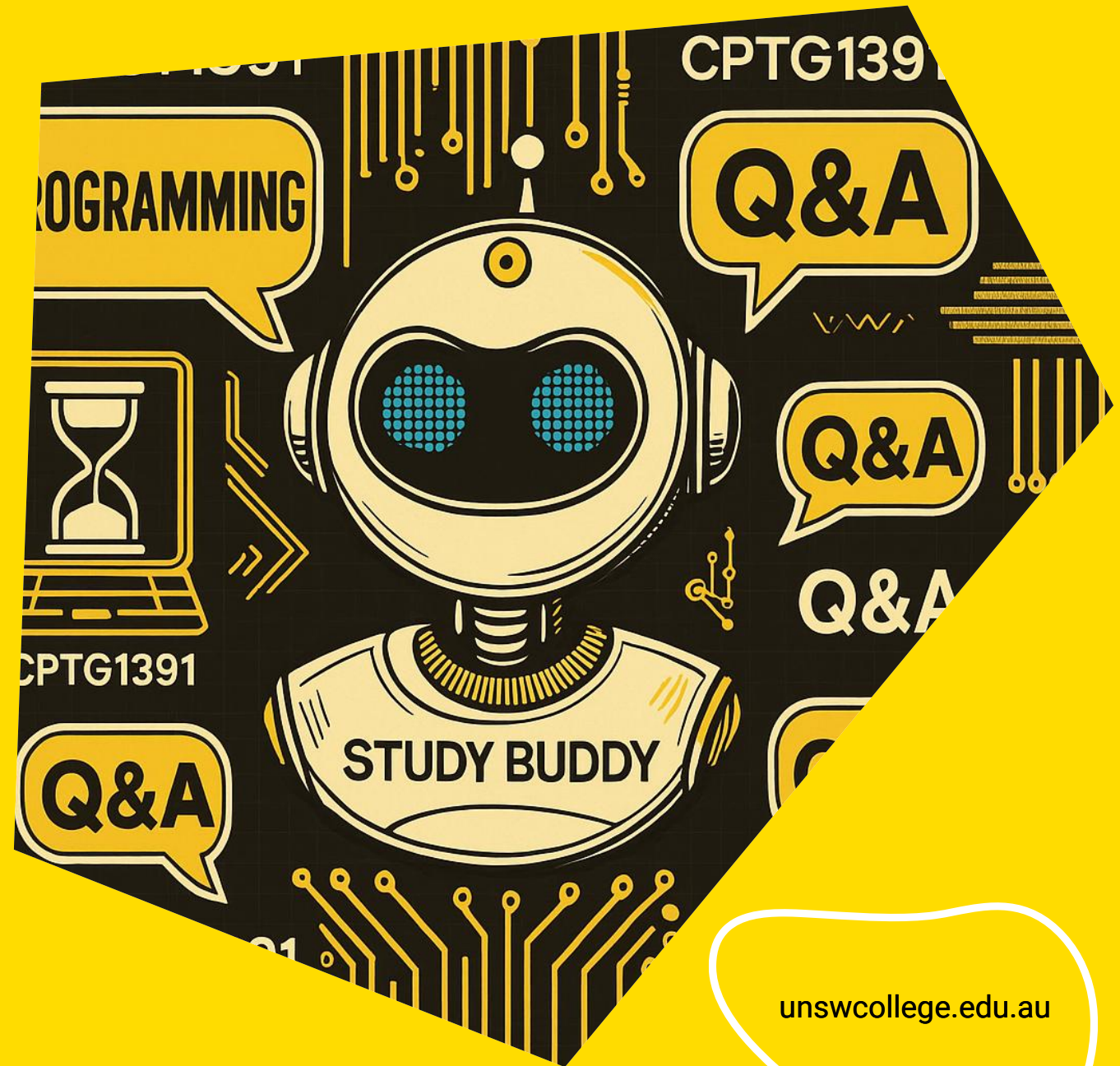
Introduction to Programming

Week 3 – Lecture 2

Lecturer and Course Convener:

Dr Pantea Aria

Static arrays



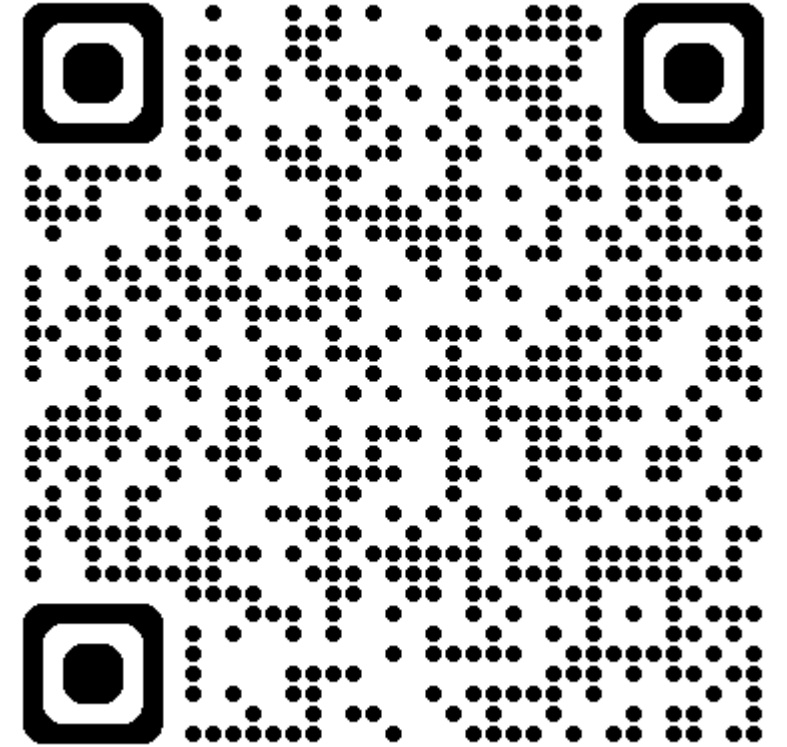
Agenda

- Last lecture
 - Functions

- Today
 - Static Arrays

Link to the Live Lecture Code

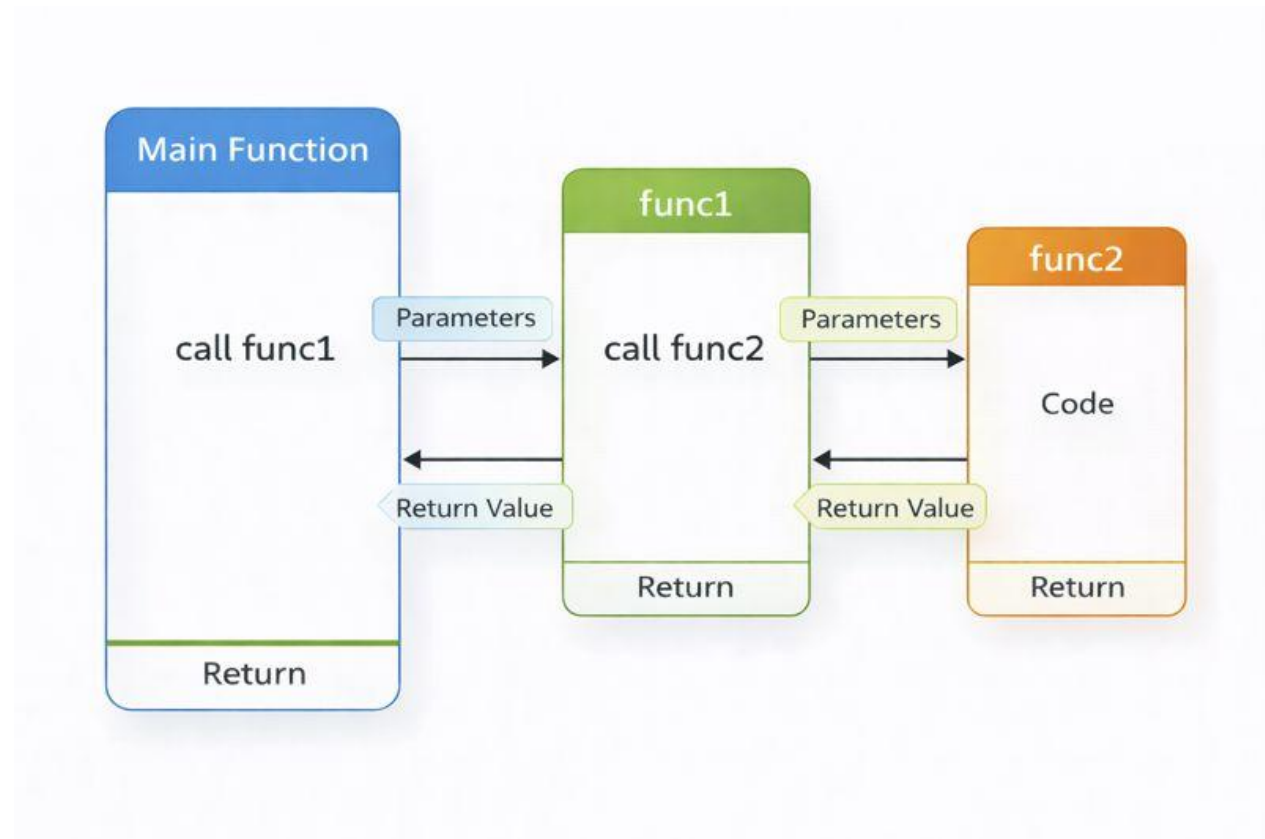
<https://cgi.cse.unsw.edu.au/~dp1091/26T2/code/03>



Live lecture code is written for teaching, not perfection.
It may include extra comments and may not always follow
ideal coding style.

Functions recap

- Write once, use many times
- You can call the same function whenever you need it
- Variables in a function stay inside that function only



Example:

Factorial Function

Writing this code every time we need a factorial would be inconvenient.

```
// calculate factorial of n
int factorial(int n) {
    int result = 1;
    int i = 1;

    while (i <= n) {
        result = result * i;
        i++;
    }

    return result;
}
```

Things to remember:

Function prototype

Call the function

Function definition (body)

```
1 #include <stdio.h>
2
3 int factorial(int n);
4
5 int main(void) {
6     int number;
7     int fact;
8
9     printf("Enter a positive integer: ");
10    scanf("%d", &number);
11
12    fact = factorial(number);
13
14    printf("Factorial of %d is %d\n", number, fact);
15
16    return 0;
17 }
18
19 // function to calculate factorial
20 int factorial(int n) {
21     int result = 1;
22     int i = 1;
23
24     while (i <= n) {
25         result = result * i;
26         i++;
27     }
28
29     return result;
30 }
```



Demo

→function_recap.c

→function_recap_int.c

Arrays

So far, we have used **one variable to store one value.**

This works fine when we only have a small number of values.

But what if we want to store many related values?

Example: Daily Temperature

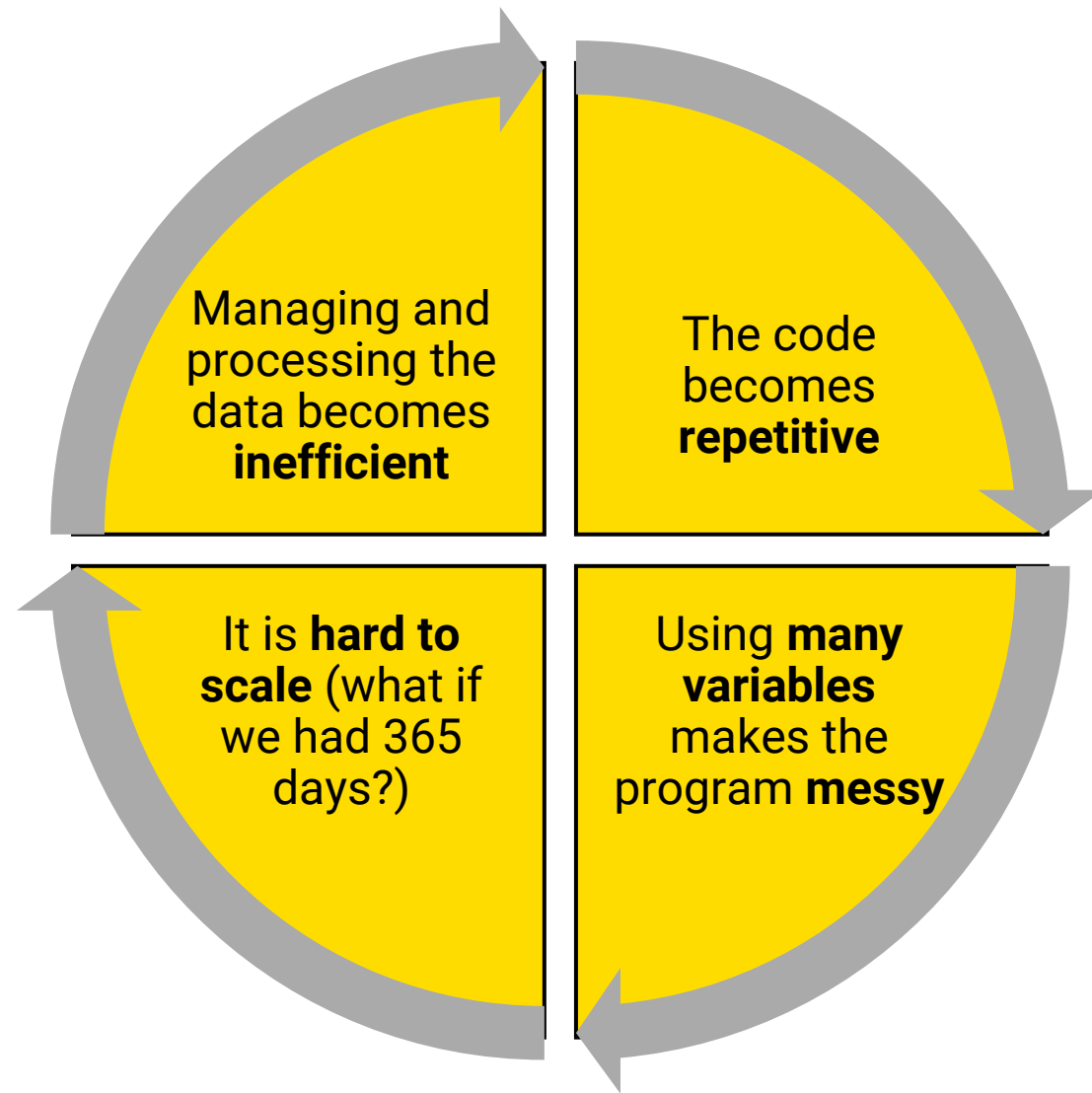
Suppose we record the temperature for each day of the week:

```
int mon = 22;  
int tue = 24;  
int wed = 23;  
int thu = 26;  
int fri = 28;  
int sat = 25;  
int sun = 21;
```

Now imagine we want to check if any day is **hotter than 25 degrees**:

```
if (mon > 25) {  
    printf("Hot day\n");  
}  
if (tue > 25) {  
    printf("Hot day\n");  
}  
// ... and so on
```

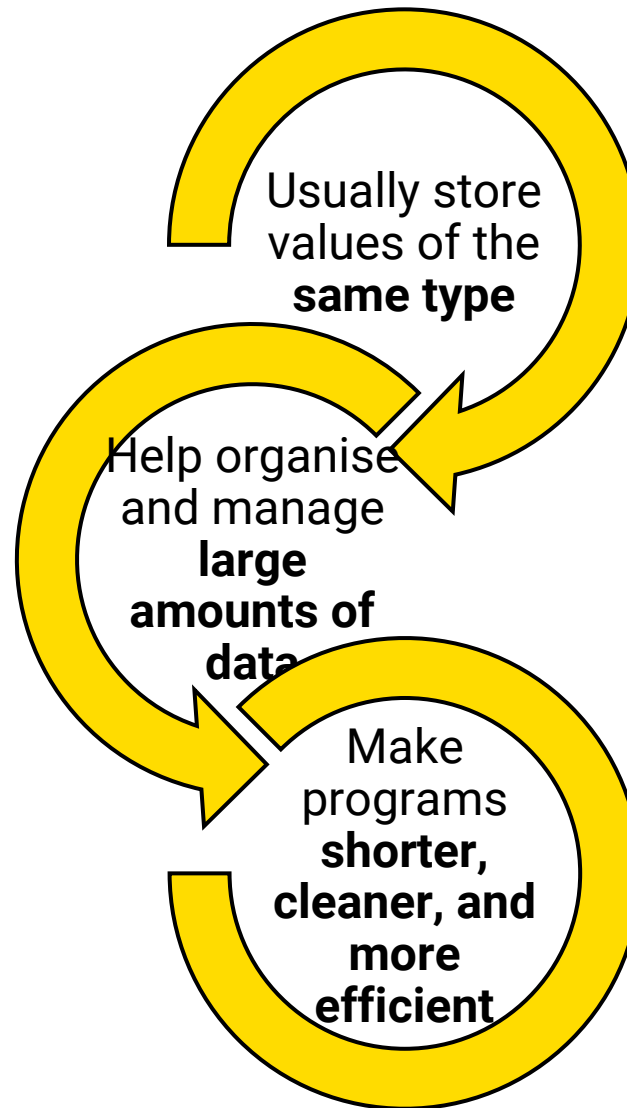
Problem with this approach is:



→ ***There must be a better way!***

Data Structures

In this course, data structures:

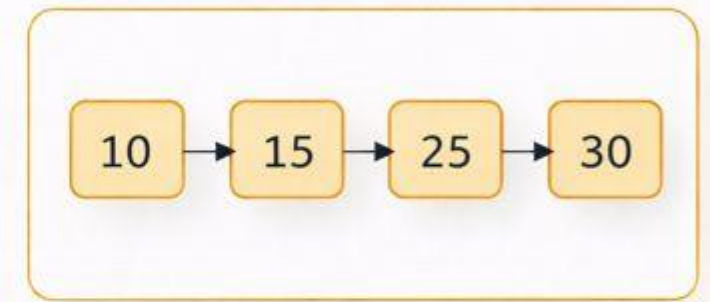


Data structures we will use in our course:

Arrays (now)



Linked lists (later)



These are fundamental and powerful tools you will use throughout your programming career.

Arrays

- C array is a collection of variables called array elements.
- All array elements must be the same type.
- Array elements don't have a name
- Array elements accessed by a number called the array index.
- Valid array indices for array with n elements are $0 \dots n - 1$
- Array can have millions/billions of elements.

You should specify the size of the Static Array



This array can hold 5 values
Size of this array is 5

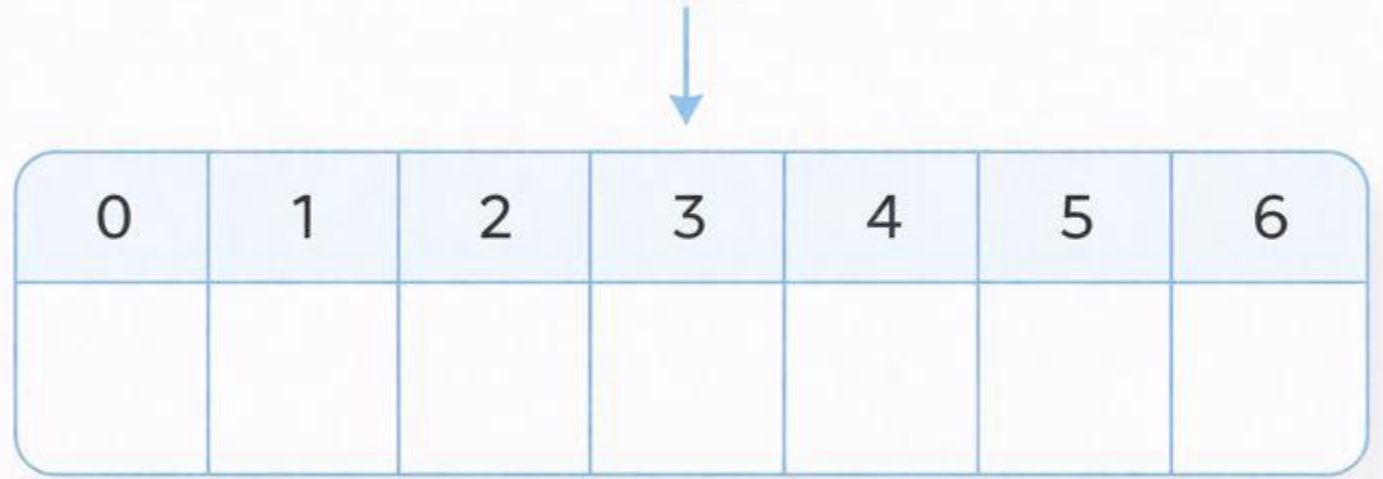
int array

index:	0	1	2	3	4
values:					

- The array contains **5 elements of type int**.
- With **each int using 32 bits**, the array requires **$5 \times 32 = 160$ bits of memory**.

Array Declaration

```
int temperature_per_day[7];
```



Declare and initialise

→ You can initialise the array while declaration:

```
int temperature_per_day[7] = {22, 24, 23, 26, 28, 25, 21};
```

index	0	1	2	3	4	6
value	22	24	23	26	25	21

→ You can initialise all elements to 0:

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

index	0	1	2	3	4	5	6
value	0	0	0	0	0	0	0

Accessing the elements

```
int temperature_per_day[7];

// Assign values to elements
temperature_per_day[1] = 30;
temperature_per_day[2] = 28;
temperature_per_day[6] = 25;

// Use the values of elements
int today_temp = temperature_per_day[0];
printf("Today's temperature is %d\n", today_temp);

// Modify an existing element
temperature_per_day[0] = temperature_per_day[0] + 1;

// Use array elements in expressions
int difference = temperature_per_day[0] - temperature_per_day[1];

// check for hot days
if (temperature_per_day[0] > 30) {
    printf("It's a hot day!\n");
}
```

Reading Arrays

→ Scanf can't read an entire array.

→ You must read the elements one by one:

```
scanf("%d", &temperature_per_day[0]);  
scanf("%d", &temperature_per_day[1]);  
scanf("%d", &temperature_per_day[2]);  
scanf("%d", &temperature_per_day[3]);  
scanf("%d", &temperature_per_day[4]);  
scanf("%d", &temperature_per_day[5]);  
scanf("%d", &temperature_per_day[6]);
```

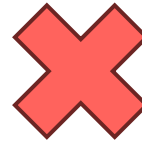
→ ***Is this a good idea??***

Arrays and Loops

```
int i = 0;

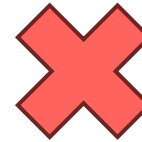
while (i < 7) {
    printf("Enter temperature for day %d: ", i);
    scanf("%d", &temperature_per_day[i]);
    i++;
}
```

// Common mistakes



// 1. Forgetting &

scanf("%d", temperature_per_day[i]); // WRONG



// 2. Going out of bounds

while (i <= 7) // WRONG

Printing Arrays

→ printf can't print an entire array.

→ You must print the elements one by one:

```
printf("%d", temperature_per_day[0]);  
printf("%d", temperature_per_day[1]);  
printf("%d", temperature_per_day[2]);  
printf("%d", temperature_per_day[3]);  
printf("%d", temperature_per_day[4]);  
printf("%d", temperature_per_day[5]);  
printf("%d", temperature_per_day[6]);
```

→ ***Is this a good idea??***

Arrays and Loops

```
int i = 0;

while (i < 7) {
    printf("Temperature for day %d: %d\n", i, temperature_per_day[i]);
    i++;
}
```

What happens step by step

- *i* starts at 0
- While *i* < 7:
- The value at `temperature_per_day[i]` is **read**
- `printf` prints the index and its value
- *i* is incremented
- Loop stops after index 6

// Common mistakes

// Using & with printf

`printf("%d\n", &temperature_per_day[i]);` // WRONG

// Going out of bounds

`while (i <= 7)` // WRONG



Demo

→array.c

Arrays and Functions

- Arrays can be **passed** into functions just like other variables.
- When **passing an array**, the function must also be told **how many elements** the array contains.
- This allows the function to safely work with **arrays of different sizes**.

```
// Function that works with an array of integers of any length  
void display_array(int length, int numbers[]);
```

Example

```
void display_array(int length, int numbers[]);

int main(void) {
    int temperatures[] = {18, 21, 25, 19};
    int scores[] = {100, 75};

    display_array(4, temperatures);
    display_array(2, scores);

    return 0;
}

void display_array(int length, int numbers[]) {
    int i = 0;
    while i < length) {
        printf("%d ", numbers[i]);
        i++;
    }
}
```

Important Behaviour of Arrays in Functions

When an array is passed to a function, **the entire array is not copied.**

The function works with the **original array** created in the caller function.

Any changes made inside the function affect the array outside the function.

This behaviour will be explained in more depth in **later weeks.**

Initialising an Array Using a Function

- A **function** can be used to **fill an array** with values.
- The **array is created in main** and **passed** into helper functions.

```
#define SIZE 5

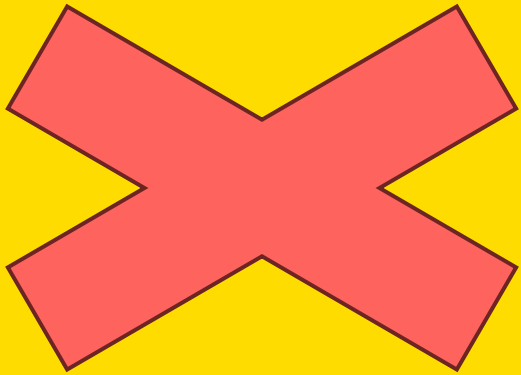
int main(void) {
    int values[SIZE];

    read_values(SIZE, values);
    display_array(SIZE, values);

    return 0;
}

void read_values(int length, int array[]) {
    int i = 0;
    while (i < length) {
        scanf("%d", &array[i]);
        i++;
    }
}
```

Returning Arrays from Functions



- Trying to **return an entire array** from a function **does not work**.
- Although the code **may compile**, it leads to serious **runtime problems**.
- We will explore why this happens later in the course.

```
// This does NOT work
int[] read_values(void) {
    int array[SIZE];

    for (int i = 0; i < SIZE; i++) {
        scanf("%d", &array[i]);
    }

    return array;    // X invalid
}
```

Demo

→array_function.c

What did we learn today?

Topic	Summary	Example
What is an Array?	An array is a collection of variables of the same type stored under one name. Each value is called an element.	<code>int marks[5];</code>
Why Use Arrays?	Arrays help store many related values efficiently instead of creating many separate variables.	Temperatures for 7 days
Array Indexing	Elements are accessed using indices starting from 0. For an array of size n , valid indices are 0 to $n-1$.	<code>marks[0], marks[4]</code>
Array Declaration	Declares an array with a fixed size.	<code>int temperatures[7];</code>
Array Initialization	Values can be assigned when the array is declared.	<code>int nums[5] = {1, 2, 3, 4, 5};</code>
Initialize All to Zero	Using <code>{0}</code> sets all elements to zero.	<code>int nums[5] = {0};</code>
Accessing Elements	Read or modify individual elements using their index.	<code>nums[2] = 10;</code>
Reading Arrays	<code>scanf()</code> reads one element at a time, usually inside a loop.	<code>scanf("%d", &nums[i]);</code>
Printing Arrays	<code>printf()</code> prints one element at a time, usually inside a loop.	<code>printf("%d\n", nums[i]);</code>
Arrays and Loops	Loops are the best way to process all elements.	<code>while (i < size)</code>
Common Mistakes	Forgetting <code>&</code> in <code>scanf()</code> Using invalid indices • Going out of bounds	<code>while (i <= size) ❌</code>
Arrays and Functions	Arrays can be passed to functions along with their size.	<code>display_array(size, arr);</code>
Important Behaviour	Arrays are not copied when passed to functions	Changes inside the function affect the original array.
Returning Arrays	Functions cannot return entire arrays directly.	<code>return arr; ❌</code>

Voice of the Student

Anonymous ongoing feedback
Anything you wanted to share with me



Voice of the Student



[Voice of the Student – Fill out form](#)

See you soon ...