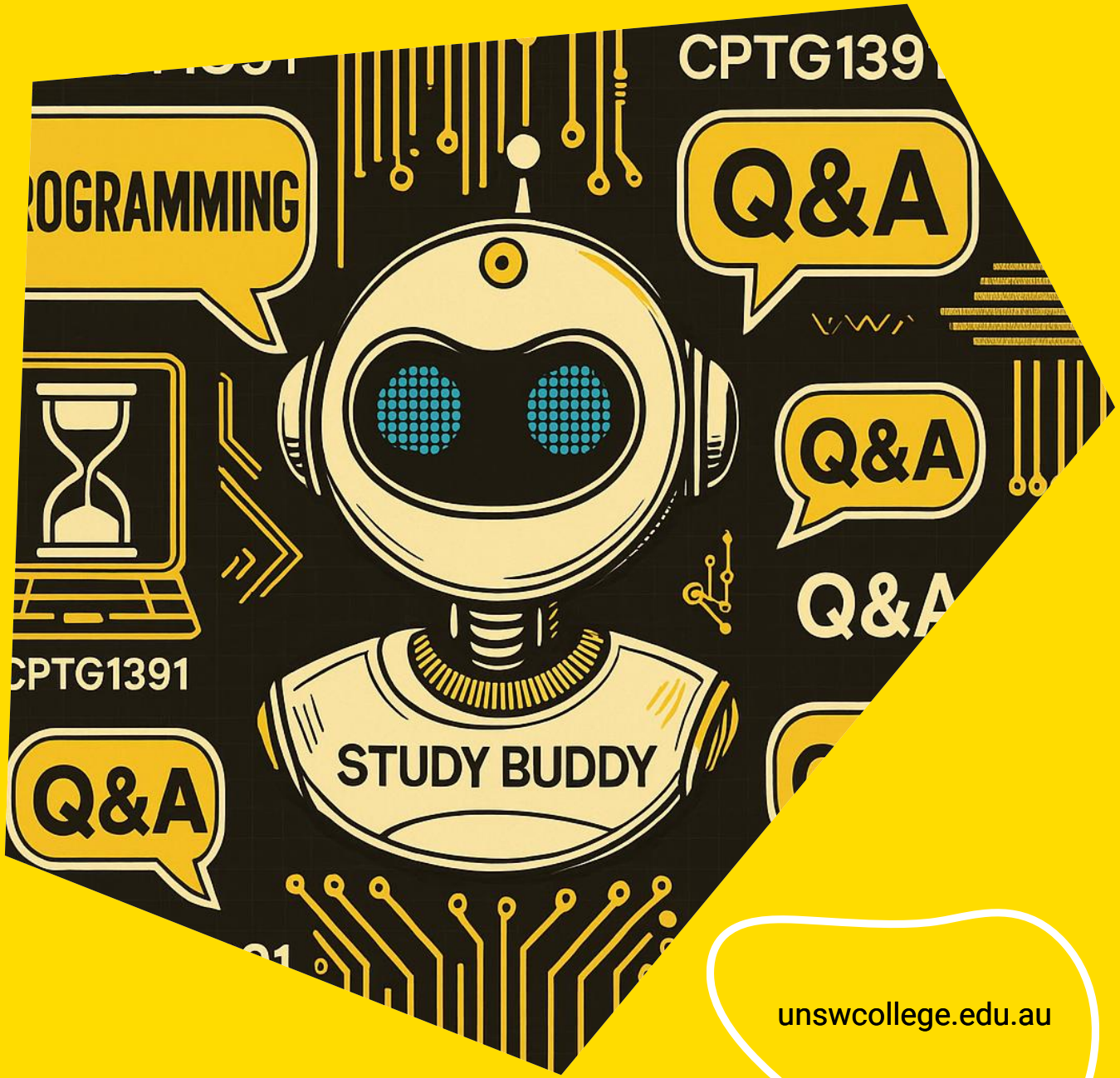
DPST1091 / CPTG1391
Introduction to Programming
Week 3 – Lecture 1

Lecturer and Course Convener:

Dr Pantea Aria

UNSW College

Functions



Agenda

- **Last lecture**
  - Nested Loops
  - Custom Data Types

- **Today**
  - Functions

Nested Loops recap

⟶A **while loop inside another while loop** is called a **nested loop.**

⟶This structure is very useful for processing **two-dimensional data**, such as **grids and matrices**.

```
     Column →   1   2   3   4   5
Row 1           1   2   3   4   5
Row 2           1   2   3   4   5
Row 3           1   2   3   4   5
Row 4           1   2   3   4   5
Row 5           1   2   3   4   5
```

Think of the grid like a **table** made of **rows** and **columns**.

⟶ **Rows** go **across** (left to right).

⟶ **Columns** go **down** (top to bottom).

Full program:

```
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
```

```c
#include <stdio.h>

#define ROWS 5
#define COLUMNS 5

int main() {
    int i = 0;

    while (i < ROWS) {
        int j = 1;
        while (j <= COLUMNS) {
            printf("%d ", j);
            j++;
        }
        printf("\n");
        i++;
    }

    return 0;
}
```

Demo

→Nested_loops_recap.c

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

UNSW College

Structs recap

→ A **struct** is a user-defined data type that groups different kinds of data together in a single unit.

→ Each field in a struct has its **own space in memory** and can be accessed individually.

→ Structs are useful when you want to store information as a **well-organised record**, such as details about a person, a student, or a product.

```
struct product {
    int id;
    double price;
};
```

Enums recap

⟶**enum** (enumeration) is a user-defined data type in C that consists of **named integer constants.**

⟶Gives **meaningful names to numbers**, improving code readability.

```
enum Weekday { MONDAY,
               TUESDAY,
               WEDNESDAY,
               THURSDAY,
               FRIDAY
             };
```

Demo

→enum_coffee_recap.c

→Struct_enum_book_recap.c

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

UNSW College


IT'S BREAK TIME!
#include <stdio.h>
#define ON_BREAK 1
int main(){
// Time for a 10 minute break! Switch to PARTY_MODE
#define PARTY_MODE ON_BREAK
if (PARTY_MODE == ON_BREAK) ;
print("Program will resume in 10 minutes...");
sleep(600);
// Take a break
exit(0);
}
10 MINUTES BREAK!
BRB
CHIPS
Relax... We'll be back soon!

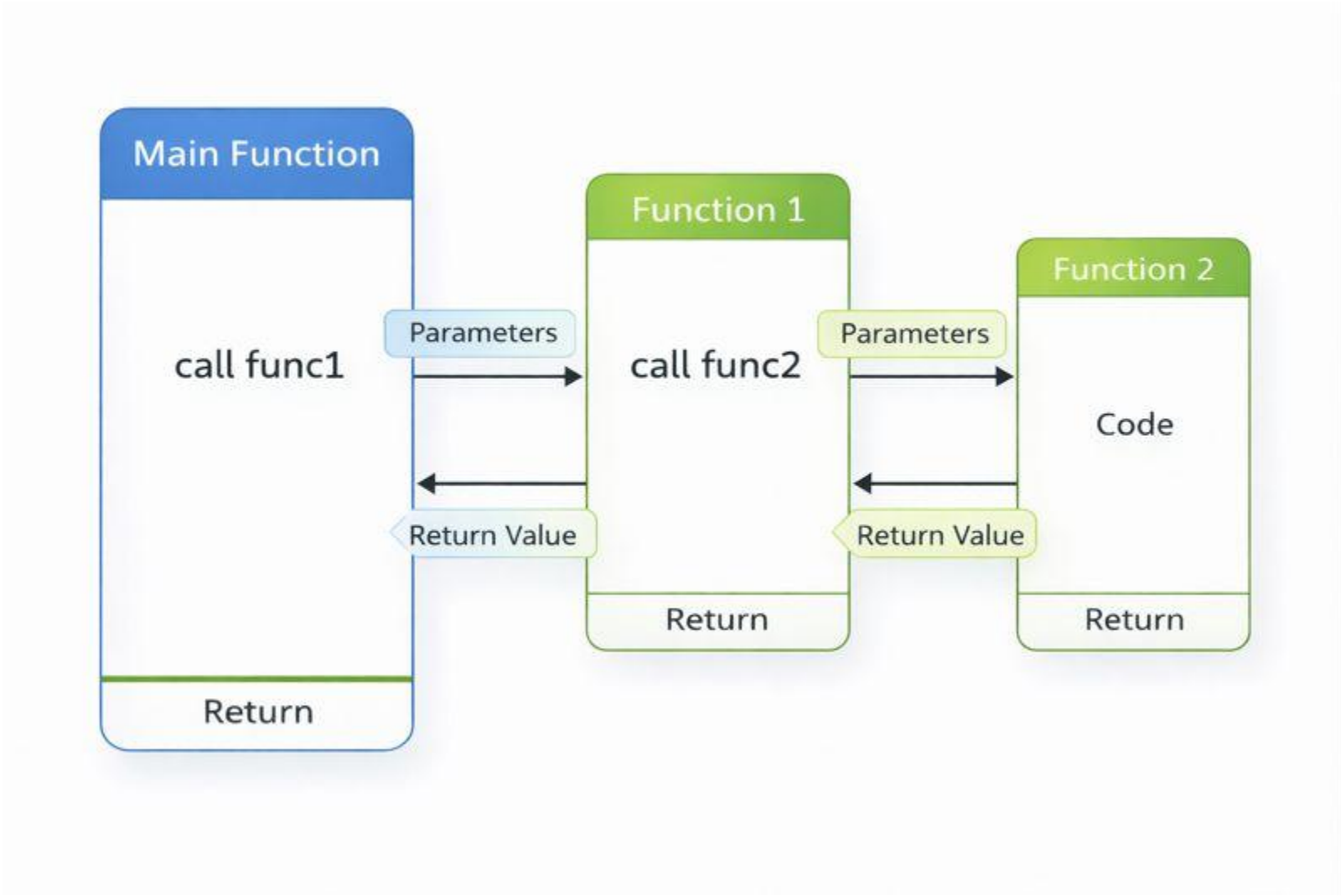
Functions

Functions allow you to:

- separate out “**encapsulate**" a piece of code serving a single purpose
- **test** and verify a piece of code
- **reuse** the code
- **shorten** code resulting in easier modification and debugging

Functions we already use: ***main(), printf(), scanf()***

How it works in a simple diagram



Demo

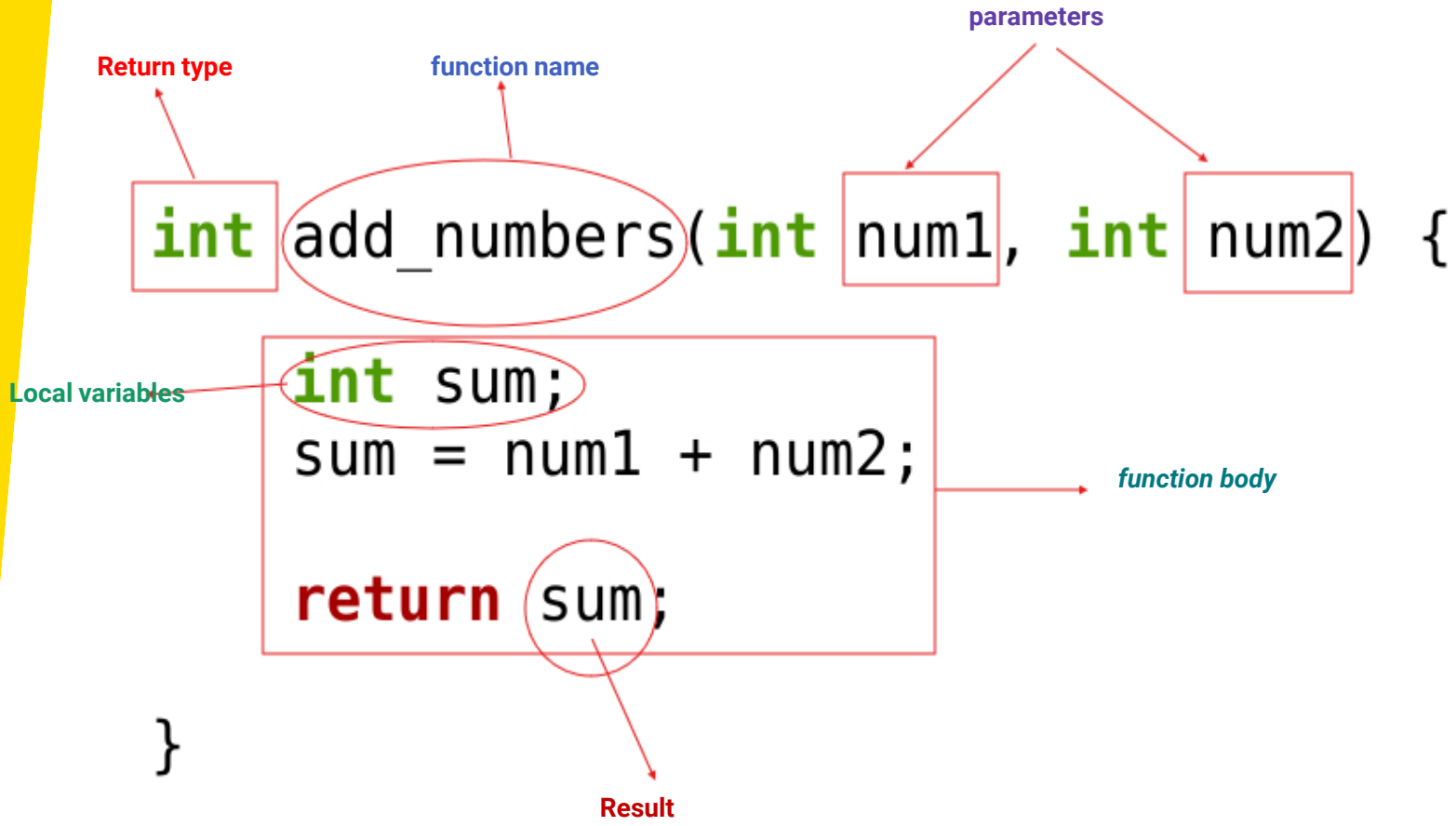
→function_hello.c

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

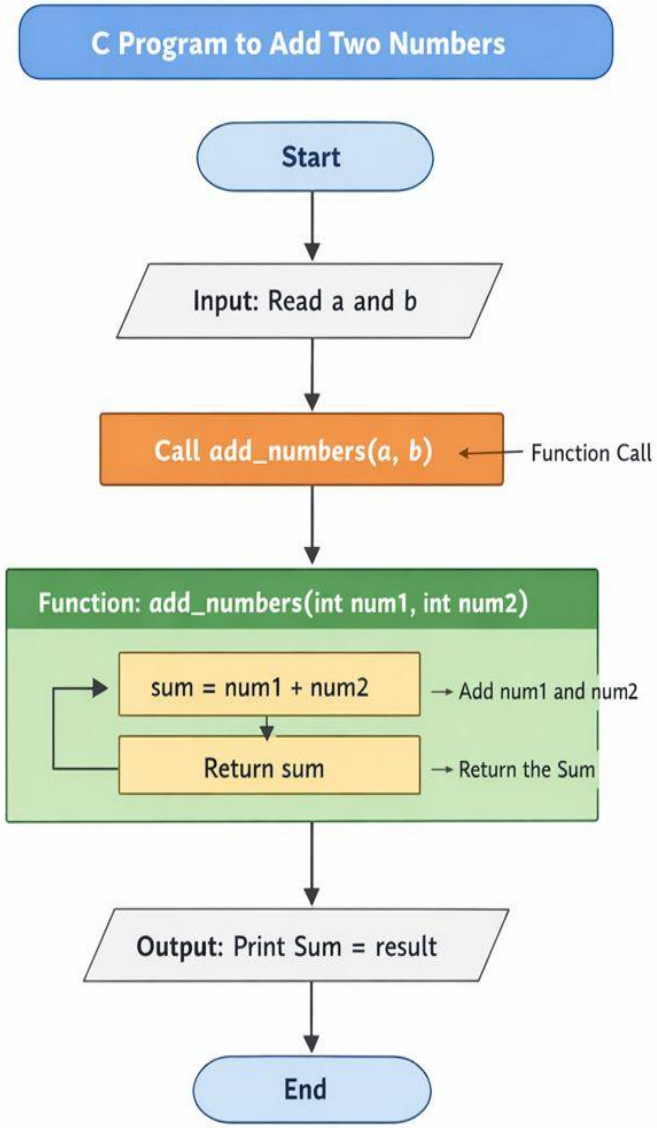
UNSW College

Structure of a Function

1. Return type
2. Function name
3. Parameters (inside brackets, comma separated) – input
4. Variables (only accessible within the body of function)
5. Return statement – output/results



Flowchart of Calling a function



The return Statement

- when a **return statement is executed**, the function **terminates**.
- the returned expression will be evaluated and, if necessary, **converted** to the type expected by the calling function.
- all **local variables and parameters** will be **thrown away** when the function **terminates**.
- functions can be declared as returning **void**, which means that **nothing is returned**. A **return** without a value statement can still be used to terminate such a function.

Function Properties

- function have a **type** - the type of the value they **return**
- type **void** for functions that return **no value**
- function can not return **arrays** (we will see later)
- function have their own **variables** created when function called and **destroyed** when function returns
- **return** statement **stops** execution of a function
- **return** statement specifies value to return, unless function is of type **void**
- **run-time error** if end of **non-void function** reached **without return**

A full Program:

```c
#include <stdio.h>

// Function prototype
double cube(double x);

int main(void) {
    double a, b;

    printf("42 cubed is %lf\n", cube(42.03));

    a = 2;
    b = cube(a);

    printf("2 cubed is %lf\n", b);

    return 0;
}

// calculate x to the power of 3
double cube(double x) {
    double result;
    result = x * x * x;
    return result;
}
```

Calling function cube

Functions with No Return Value (called procedures too)

Some functions do not to compute a value.
They are useful for "**side-effects**" such as output.

```
void print_sign(int b) {
    if (b < 0) {
        printf("negative");
    } else if (b == 0) {
        printf("zero");
    } else {
        printf("positive");
}
```

Demo

→function_void.c

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

UNSW College

Function Parameters

- functions take **0 or more** parameters
- parameters are variables **created each time function** called and destroyed when function returns
- C functions are ***call-by-value*** (but beware arrays)
- parameters initialized with the value **supplied by the caller**
- if parameters **variables changed** in the function **has no effect outside the function**

```c
#include <stdio.h>

// Function prototype
void change_values(int x, int y);

int main(void) {
    int a = 5;
    int b = 10;

    printf("Before function call: a = %d, b = %d\n", a, b);

    // Call-by-value: copies of a and b are passed
    change_values(a, b);

    // Original variables are unchanged
    printf("After function call: a = %d, b = %d\n", a, b);

    return 0;
}

// Function definition (after main)
void change_values(int x, int y) {
    // Parameters are created when the function is called

    x = x + 10;
    y = y + 20;

    printf("Inside function: x = %d, y = %d\n", x, y);

    // Parameters are destroyed when the function returns
}
```

Function Prototypes

Specifies key information about the function:

- function return type
- function name
- number and type of function parameters

Allows top-down order of functions in file More readable
Allows us to have function definition in separate file.

Example prototypes:

```
double power(double x, int n);
void print_sign(int b);
```

Demo

→function_add.c

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

UNSW College

Library functions

⟶ Over 700 function are defined in the C standard library.

⟶ You'll need to use less than 20 of these in this course.

⟶ The C compiler needs to see a prototype for these functions before you use them.

⟶ You do this indirectly with **#include** line

⟶ For example, **stdio.h** contains prototypes for **printf** and **scanf**

```c
#include <stdio.h>

int main(void) {

    printf("I love Programming!\n");

}
```

Example:

```c
#include <stdio.h>
#include <math.h>

int main(void) {
    double number = 9.0;
    double result1, result2;

    // Using sqrt() from math.h
    result1 = sqrt(number);

    // Using pow() from math.h
    result2 = pow(number, 2);

    printf("Square root of %.2f is %.2f\n", number, result1);
    printf("%.2f squared is %.2f\n", number, result2);

    return 0;
}
```

structs and functions

A structure can be passed as a parameter to a function:

```
void print_student(student s) {

        printf("%s z%d\n", s.name, s.zid);

}
```

A function can **return a structure**:

```
student read_student(int zid, double marks) {

        student s;
        s.zid = zid;
        s.marks = marks;

        return s;

}
```

Demo

→function_struct.c

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

UNSW College

Voice of the Student

Anonymous ongoing feedback
Anything you wanted to share with me

Voice of the Student

26T1 Voice of the Student – Fill out form

See you soon …

UNSW College