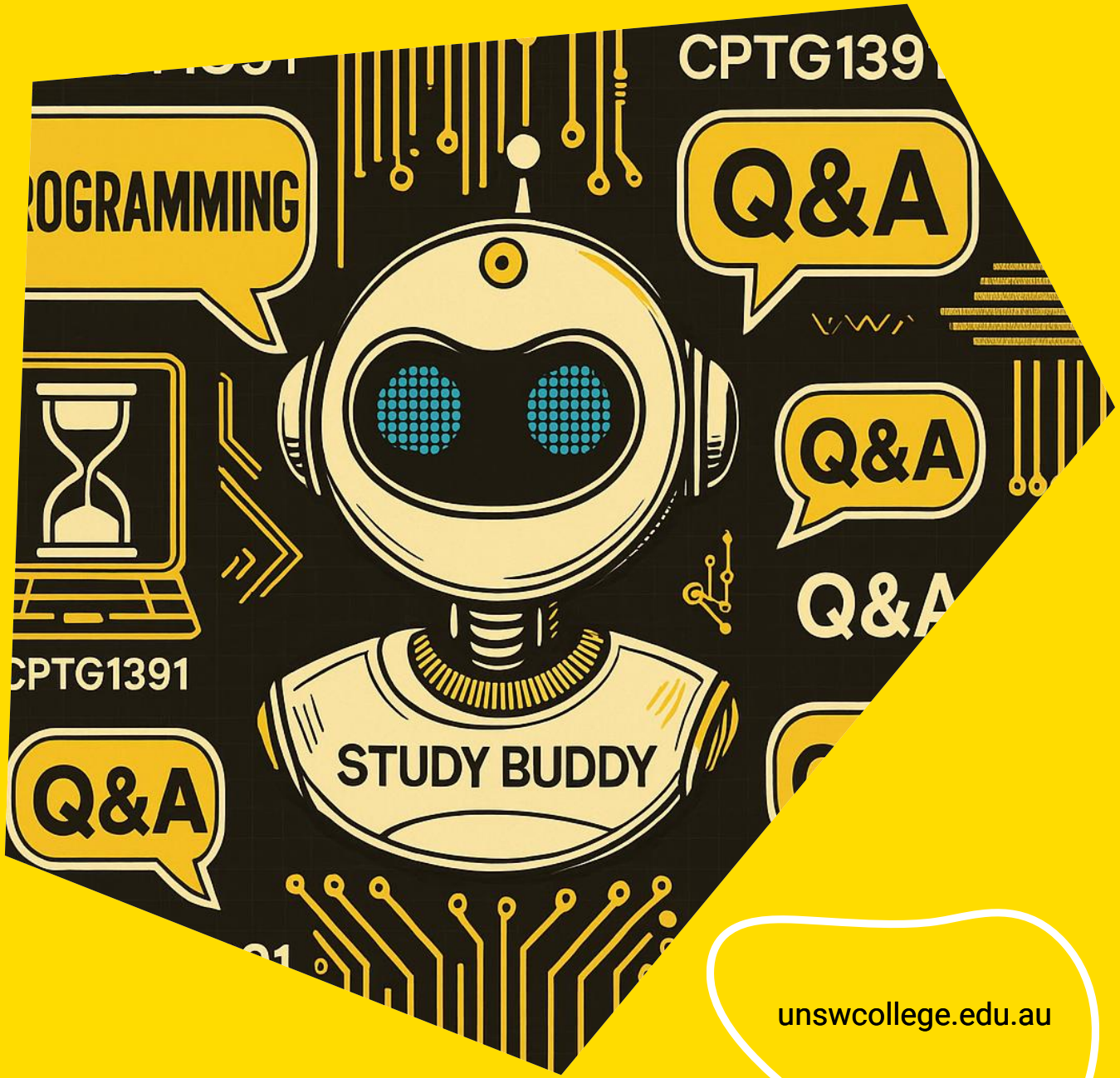
DPST1091 / CPTG1391
Introduction to Programming
Week 4 – Lecture 2

Lecturer and Course Convener:

Dr Pantea Aria

UNSW College

Strings



unswcollege.edu.au

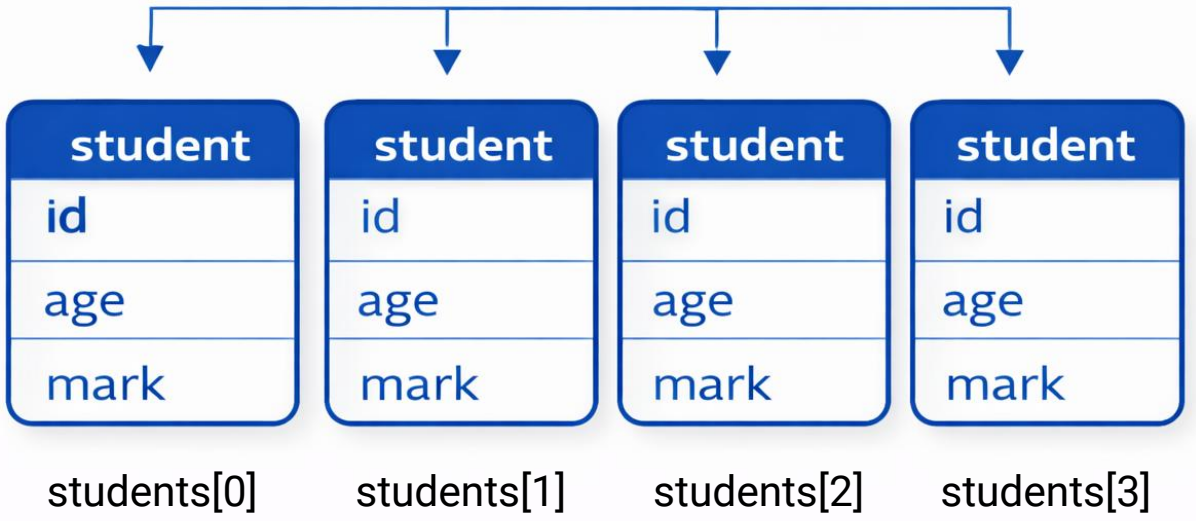
Agenda

- **Last lecture**
  - ⟶arrays of structs
  - ⟶2D arrays

- **Today**
- **strings**

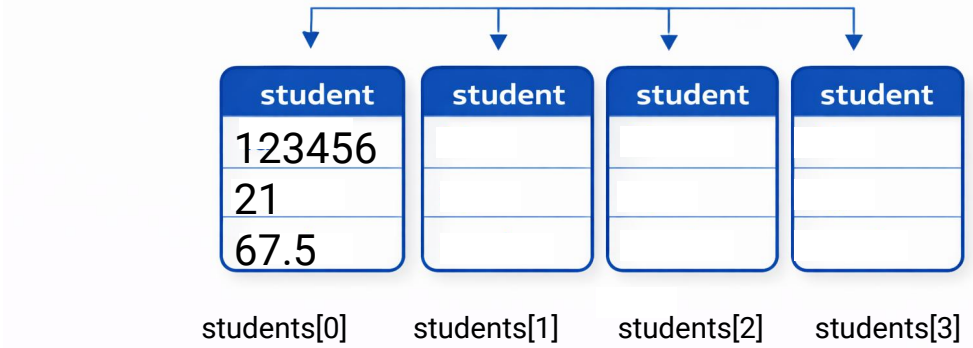
Array of Struct recap

```
struct student {
    int id,
    int age,
    double mark;
};

struct student students[4];
```



Accessing the elements of an array of structures

```
struct student {
   int id,
   int age,
   double mark;
};

struct student students[3];
```




```
students[0].id = 123456;
students[0].age = 21;
students[0].mark = 67.5;
```

2 Dimentional Arrays

arrays of arrays

It stores data in **rows and columns**, like a **table or grid**.
How to declare?

```
<type>  <identifier>[<rows>][<cols>];
```

```
// Declaring a 2D array
int marks[3][4];
```

This means:

3 rows
4 columns

Total elements = 3 × 4 = 12

You can picture it like this:

```
[][][][]
[][][][]
[][][][]
```

Initialising a two-dimensional array

```
// Initialising a 2D array
int marks[3][4] = {
                    {10, 12, 14, 16},
                    {11, 13, 15, 17},
                    {18, 19, 20, 21}
                  };
```

Each inner {} is one row.

	col 0	col 1	col 2	col 3
row 0	10	12	14	16
row 1	11	13	15	17
row 2	18	19	20	21

marks

How to traverse the array

A **2D array has two dimensions**:

- **rows**
- **columns**

So, we need:

- one loop to move through the **rows**
- one loop to move through the **columns**

```
int row = 0;
// outer loop for rows
while (row < 3) {
    int col = 0;
    // inner loop for columns
    while (col < 4) {
        printf("%d ", marks[row][col]);
        col++;
    }
    printf("\n");
    row++;
}
```

Demo

→2Darrays_recap

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!
print_table(sum);
BRB
CHIPS
Relax... We'll be back soon!

Strings

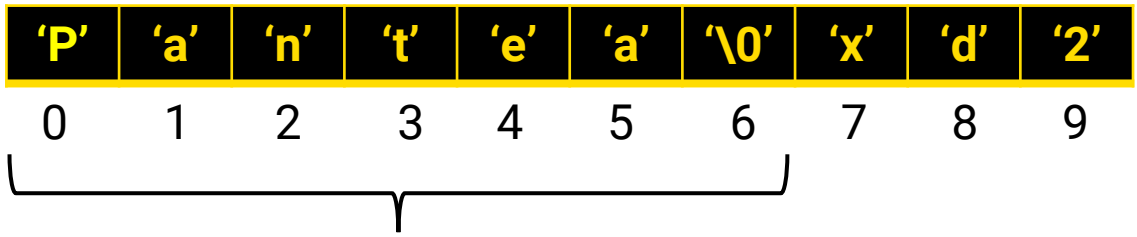
Strings represent **sequences of characters**

In C, a string is defined as: an **array** of characters (**char**)

terminated by the **null character '\0'** stored in consecutive memory locations

Null Terminator '\0'

- Every string in C **must** end with the **null terminator '\0'**
- **Without** it, the data is **not a string**, only an array of char
- The **character array** must be **large enough** to include the '\0'
- The null terminator is **not printed or visible** when the string is displayed
- It tells the program where the **string ends**, especially when **looping** through characters
- Any data stored **after '\0'** is **ignored** and not considered part of the string



How to initialise strings

- A string is stored as a **sequence of characters** in a `char` array
- Each character occupies one position in the array
- The final element is always the **null terminator** `'\0'`

```
Index:   0    1    2    3    4    5    6
Value:  'p'  'a'  'n'  't'  'e'  'a'  '\0'
```

```c
// The long (manual) way
char word[] = {'p', 'a', 'n', 't', 'e', 'a', '\0'};


// The simpler and preferred way
char word[] = "pantea";
```

Both declarations create **exactly the same** string in memory

The **string literal** version **automatically adds the '\0'**

Use **double quotes** " to wrap the string literal

Printing a string

```c
// Printing a string character by character
char name[] = "pantea";

int i = 0;
while (name[i] != '\0') {
    printf("%c", name[i]);
    i++;
}
printf("\n");
```

The loop stops at '\0'

Characters are accessed one at a time using an index

```c
// The easier way
// Using printf with %s
char name[] = "pantea";
printf("%s", name);
```

%s automatically prints characters until it reaches '\0'

Demo

→strings_intro.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

reading a string

`scanf("%s", ...)` **must not be used** to read strings.

It does **not check the size of the array**

- Input that is too long can cause a **buffer overflow**
- This may **overwrite other memory** and **crash** the program

Buffer overflows are a **serious security vulnerability**

- They can be exploited by attackers
- You will learn more about this in **DPST1092**

`%s` often behaves **differently from what is expected**

- It **stops reading when** it encounters **whitespace** (spaces, tabs, newlines)

For these reasons, `scanf("%s", ...)` is **forbidden in the style guide**

So, how do we read strings?

fgets

Strings should be read using **`fgets`**:

```
fgets(array, size, stream);
```

`fgets` parameters

•**`array`**

The character array where the string will be stored

•**`size`**

The total size of the array

- **`fgets`** reads **at most `size - 1` characters**
- This leaves space for the null terminator `'\0'`

•**`stream`**

The source of the input

- In this course, this will always be **`stdin`**
- (**`stdin`** refers to standard input from the keyboard)

How `fgets` reads input

A single call to `fgets` reads characters until **one of the following occurs**:

- `size - 1` **characters** have been read
  - This leaves room for the null terminator `'\0'`
- A **newline character** (`'\n'`) is encountered
  - The newline **is stored** in the array
- **end of file** (**eof**) is reached
  - For terminal input, this is `Ctrl + D` on a line by itself

Example

```
  char name[MAX_LENGTH];

// Read a string of up to MAX_LENGTH characters
// from standard input (the terminal)
//
fgets(name, MAX_LENGTH, stdin);
```

If `MAX_LENGTH` is **10** and the user types: Pantea
Then the name array will contain:

‘P’	‘a’	‘n’	‘t’	‘e’	‘a’	‘\n’	‘0’	?	?
0	1	2	3	4	5	6	7	8	9

Reading strings in a loop

```c
#include <stdio.h>

// The name will be at most 14 characters (+1 for '\0')
#define MAX_LENGTH 15

int main(void) {
    char name[MAX_LENGTH];

    printf("Please enter your name: ");

    // Read a line of input safely using fgets
    // This stores the input (including '\n' if present)
    while (fgets(name, MAX_LENGTH, stdin) != NULL) {
        // Each successful read replaces the contents of name
    }

    return 0;
}
```

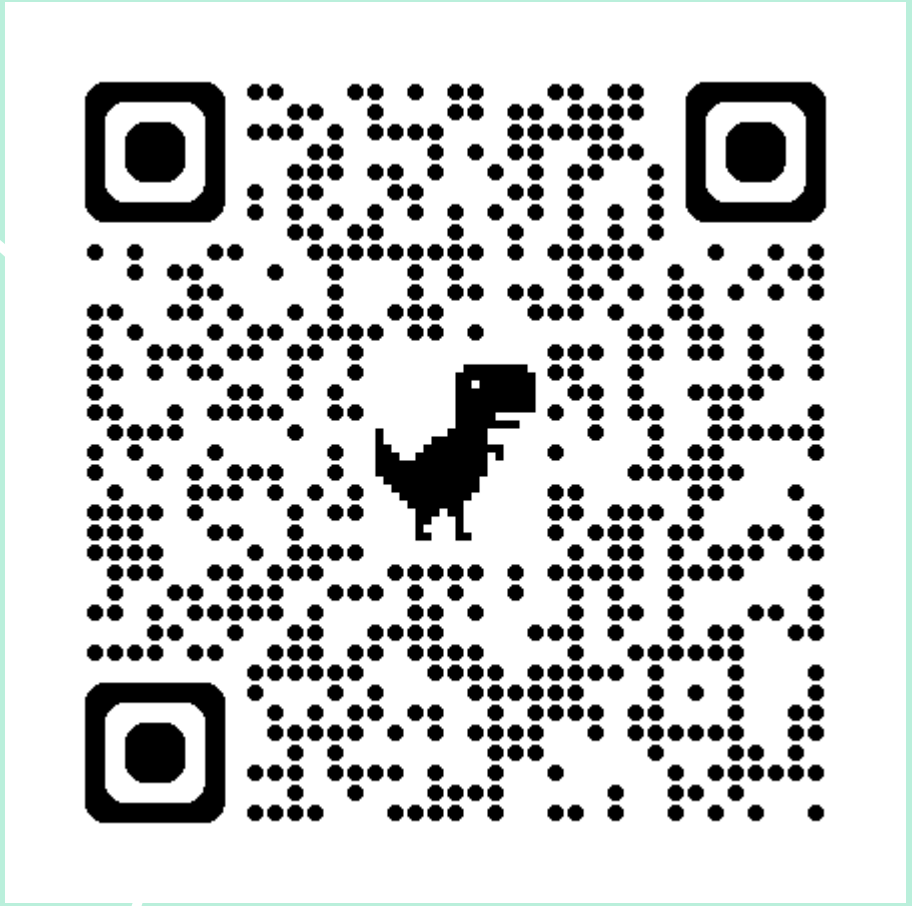
Useful library functions for chars

```
#include <ctype.h>
```

Function	What it does	Example
`toupper()`	Converts a character to uppercase	`toupper('p') → 'P'`
`tolower()`	Converts a character to lowercase	`tolower('A') → 'a'`
`isupper()`	Checks if a character is uppercase	`isupper('P') → 1`
`islower()`	Checks if a character is lowercase	`islower('p') → 1`

Demo

→strings_reading.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

Useful library functions for strings

```
#include <string.h>
```

Function	What it does	Example
`strlen()`	Returns the number of characters in a string (excluding '\0')	`strlen("pantea")`
`strcpy()`	Copies one string into another	`strcpy(dest, "pantea");`
`strcat()`	Appends one string to the end of another	`strcat(name, "123");`
`strcmp()`	Compares two strings	`strcmp("pantea", "pantea")`
`strchr()`	Finds the first occurrence of a character	`strchr("pantea", 't')`

Example

```c
 // Declare an array to store a string
char pet[MAX_LENGTH] = "Oreo";

// Copy a new string into the array
// Make sure the array is large enough to avoid overflow
strcpy(pet, "Buddy");

printf("%s\n", pet);

// Find the length of the string (does NOT count '\0')
int length = strlen(pet);
printf("%s has length %d\n", pet, length);
```

Example

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

int main(void) {
    char name1[] = "pantea";
    char name2[] = "Pantea";
    char name3[] = "sara";

    // Case 1: strings are equal
    if (strcmp(name1, "pantea") == 0) {
        printf("name1 is equal to \"pantea\"\n");
    }

    // Case 2: first string comes before second (negative result)
    if (strcmp(name1, name3) < 0) {
        printf("\"%s\" comes before \"%s\"\n", name1, name3);
    }

    // Case 3: first string comes after second (positive result)
    if (strcmp(name1, name2) > 0) {
        printf("\"%s\" comes after \"%s\"\n", name1, name2);
    }

    return 0;
}
```

Demo

→strings_functions.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