DPST1091 / CPTG1391

# Introduction to Programming

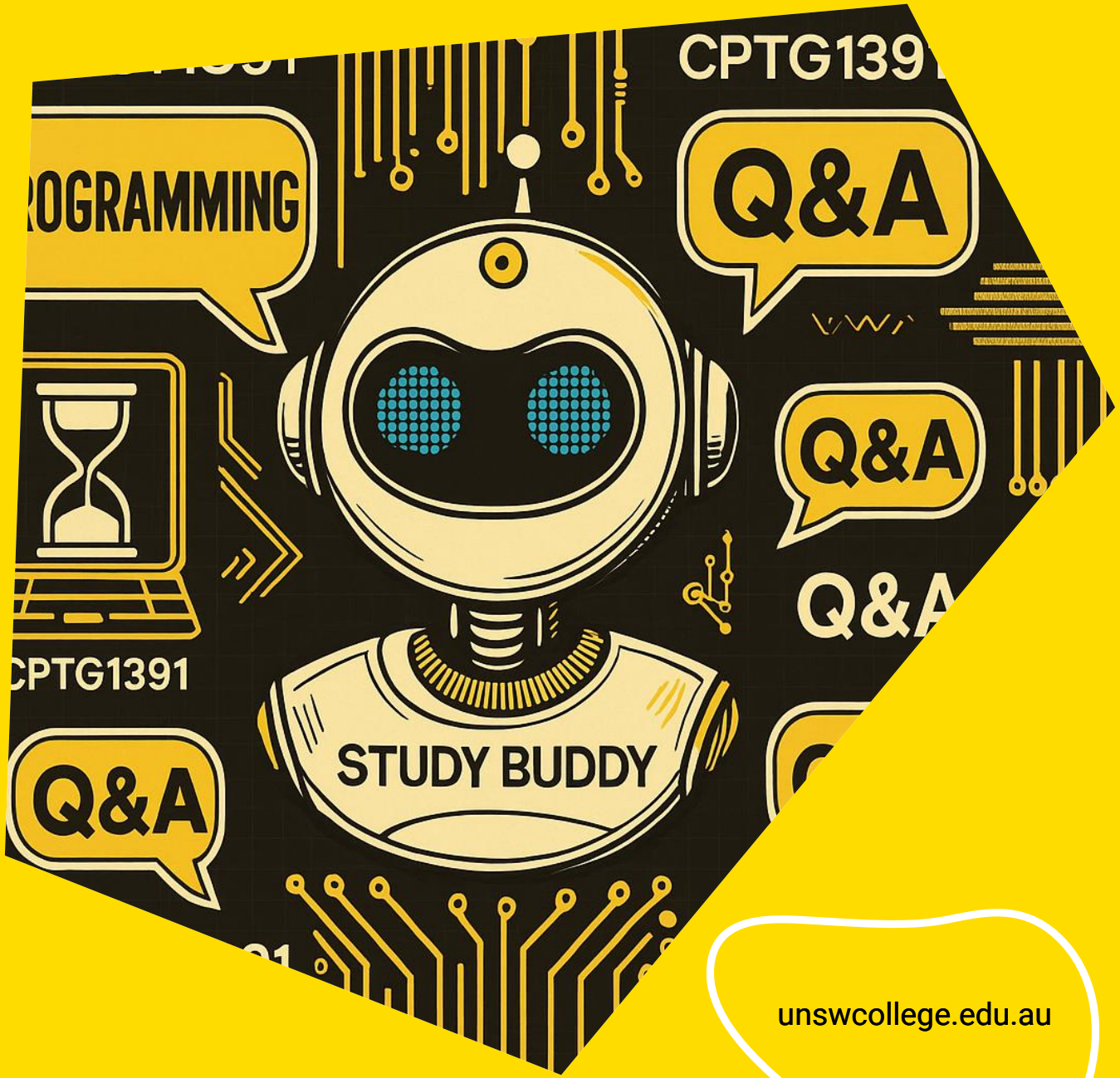
## Week 5 – Lecture 1


**Lecturer and Course Convener:**

**Dr Pantea Aria**

UNSW College

# Arrays of Strings




unswcollege.edu.au

# Agenda

- **Last lecture**
  → strings

- **Today**
- **Assignment 1 info**
- **Arrays of strings**

# Assignment 1 Overview

- **Due: Friday, Week 8** at **9:00 AM**
- This is an **individual assignment**

**Purpose of the assignment**

- Use **arrays and two-dimensional arrays** to solve a larger programming problem
- Apply **functions** to structure and organise your code
- Build **debugging skills** and persistence when resolving errors
- Practice writing code that follows **good programming style**
  - **Style is assessed** and contributes **20% of the total mark**

# Assignment1 Structure

- The assignment is divided into **stages**
  - ***Stages 1, 2, and 3*** are the core requirements
- ***Stage 4***
  - Is a **challenge extension**
  - Not all students are expected to complete it
  - High marks are achievable **without** completing this stage
  - You are **on track in the course** even if you do not reach Stage 4

# How to approach the assignment

| |
|---|
| **Read the specification carefully before starting** |
| **Download** the **starter code** and familiarise yourself with its structure |
| **Run** the **reference implementation** to understand the expected behaviour of the final program |
| **Complete** the assignment **one substage** at a time |
| **Test** each stage thoroughly **before** progressing to the next |
| If you encounter **difficulties**, seek help early from:<br>• **Tutors in labs,**<br>• **The course forum**<br>• **Consultation sessions** |

# Demo

→Assignment 1

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

UNSW College

# Strings recap

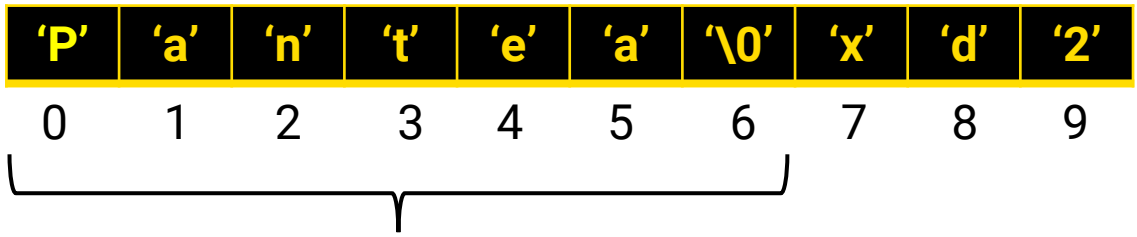
Strings represent **sequences of characters**

**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'*

# Fgets for reading a string

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

# 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` |

# 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);
```

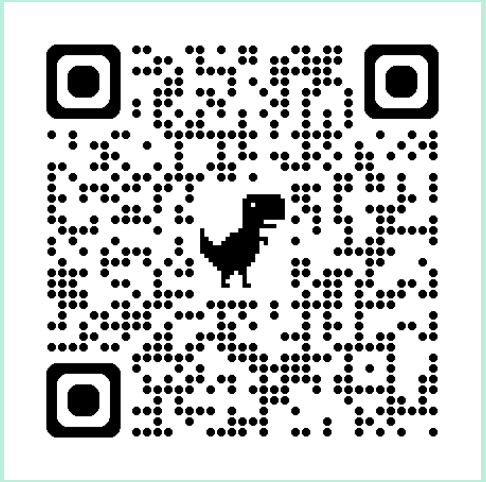
# Example

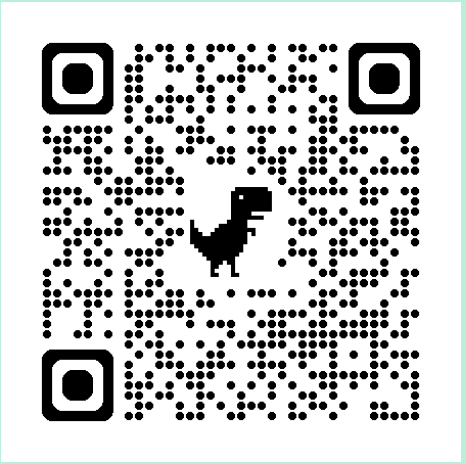
```c
#include <stdio.h>
#include <string.h>

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

→From week 4 string_reading.c
→From week 4 string_functions.c

→From week 5 strings_newline.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!
print_table(sum);
BRB
CHIPS
Relax... We'll be back soon!

# Array of Strings

An **array of strings** is a **two-dimensional array of characters**

**Rows = individual strings**

**Columns = characters within each string**

# Example

```
char names[3][10] = {"pantea", "sara", "alex"};
```

| “Pantea” | “sara” | “alex” |
|---|---|---|
| 0 | 1 | 2 |

- names can **store 3 strings**
- Each string can store **up to 9 characters plus the null terminator ‘\0’**

```
names[0] → "pantea"
names[1] → "sara"
names[2] → "alex"
```

| | col 0 | col 1 | col 2 | col 3 | col 4 | col 5 | col 6 | col 7 | col 8 | col 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| row 0 | ‘P’ | ‘a’ | ‘n’ | ‘t’ | ‘e’ | ‘a’ | ‘\0’ | | | |
| row 1 | ‘s’ | ‘a’ | ‘r’ | ‘a’ | ‘\0’ | | | | | |
| row 2 | ‘a’ | ‘l’ | ‘e’ | ‘x’ | ‘\0’ | | | | | |

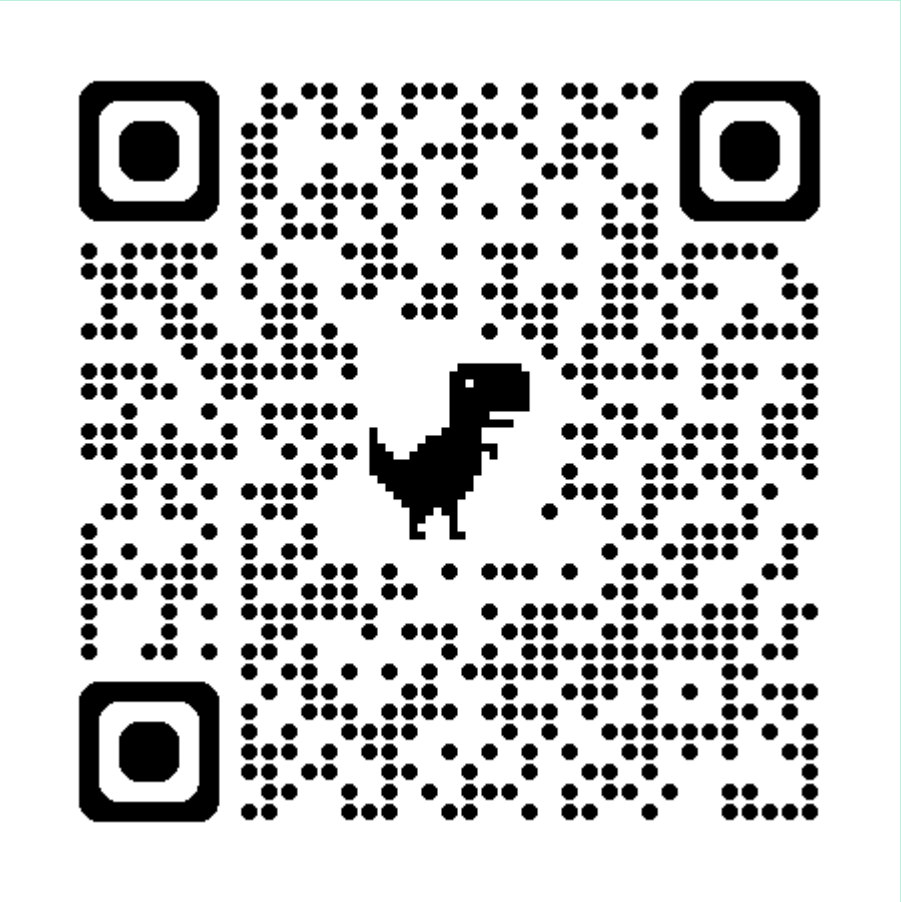
# Another example

```c
char names[3][10] = {"pantea", "sara", "alex"};

// Using one index selects a whole string (a row)
// This prints "sara"
printf("%s\n", names[1]);
```

```c
char names[3][10] = {"pantea", "sara", "alex"};

// Using two indexes selects a single character
// This prints the 'e' from "alex"
printf("%c\n", names[2][2]);
```

# Demo

→array_of_strings.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



26T1 Voice of the Student – Fill out form

# See you soon …

UNSW College