

DPST1091 / CPTG1391

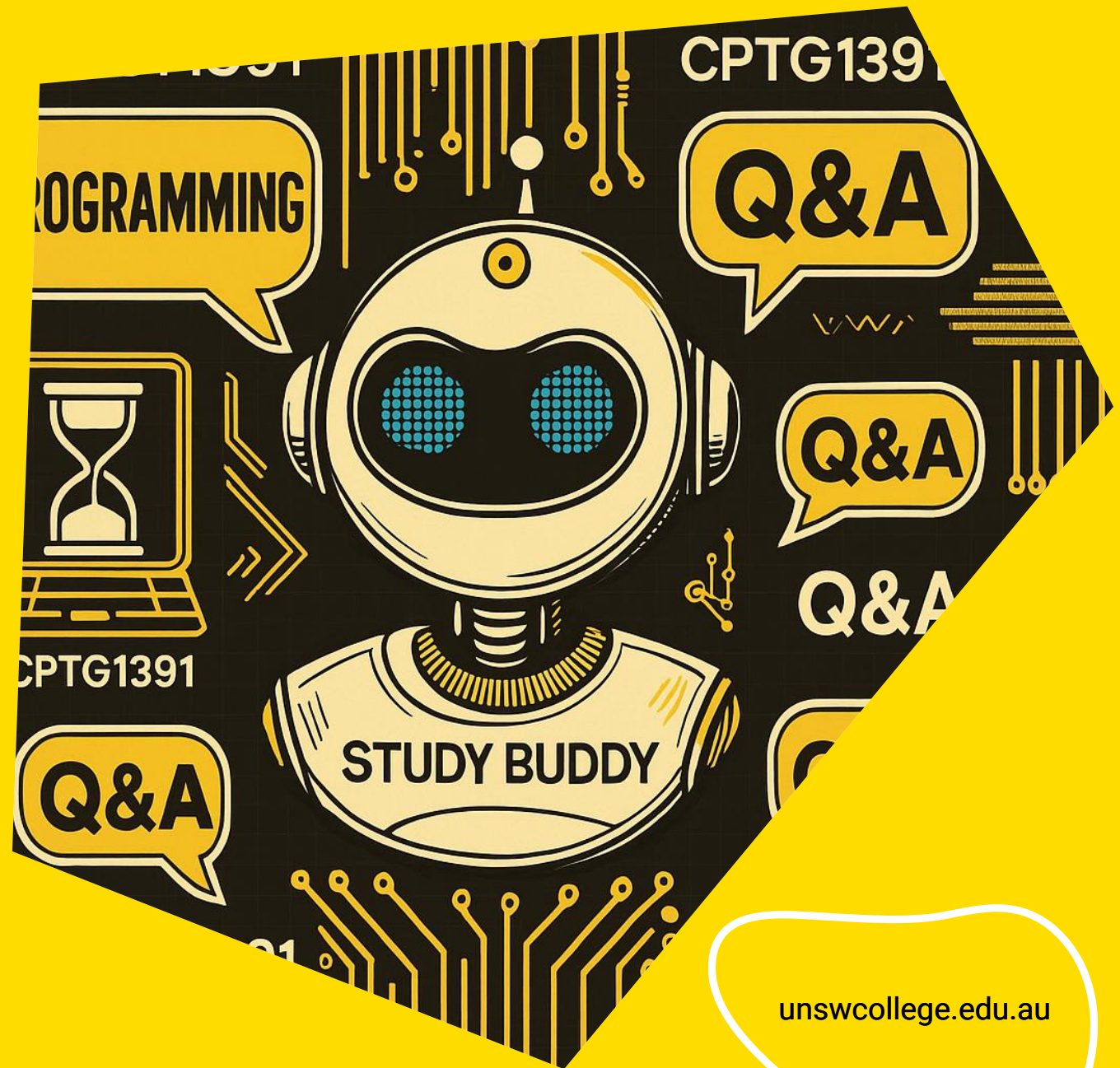
Introduction to Programming

Week 2 – Lecture 2

Lecturer and Course Convener:

Dr Pantea Aria

Custom Data Types



unswcollege.edu.au

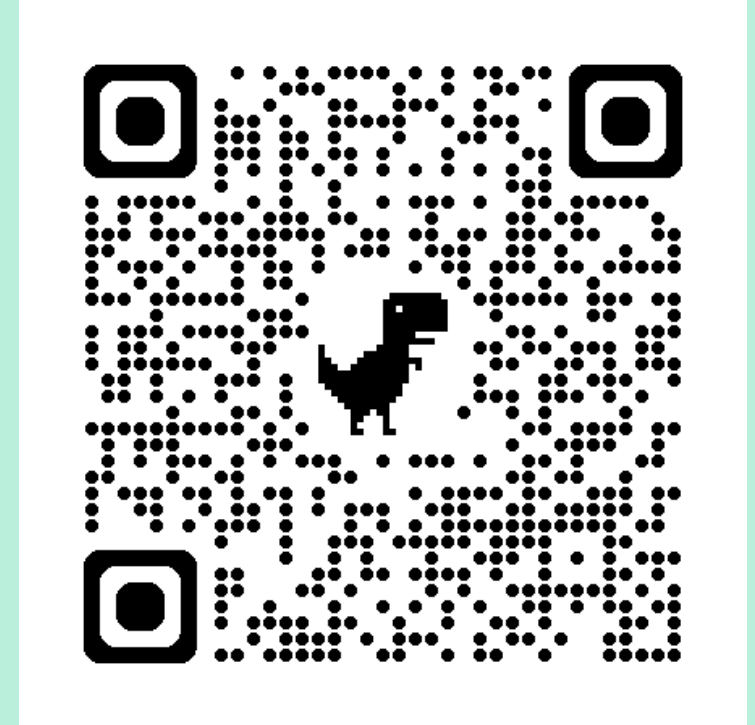
Agenda

- **Today**

- Nested Loops
- Custom Data Types

Link to the Live Lecture Code

<https://cgi.cse.unsw.edu.au/~dp1091/26T2/code/02/>



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

if statements recap

```
if(<condition>) {  
    do_if_true();  
} else if(<second_condition>) {  
    do_if_second_true();  
} else {  
    do_if_both_false();  
}
```

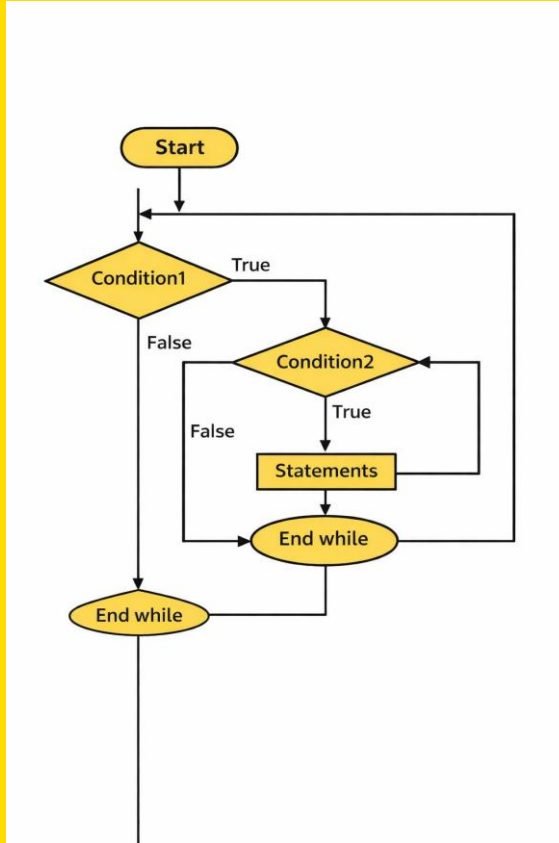
- A **condition** is a value that is either **true (1)** or **false (0)**.
- We can **compute a condition** by evaluating an expression.
- For example, *temperature < 0* will be **true (1)** if *temperature* is below zero.
- Conditions are widely used in C, such as in **if statements**, **while loops**, and other control structures.

while loops

```
while(<condition>) {  
    do_something_until_condition_is_true;  
}
```

- If the condition is **true**, execute the body of the loop.
- After the body finishes, **check the condition again**.
- If the condition is still **true**, execute the body again.
- Repeat this process until the condition becomes **false**.

Nested while loops



—→ A **while loop inside another while loop** is called a nested loop.

—→ Each time the **outer loop** executes once, the **inner loop** runs completely from start to finish.

—→ This means the **inner loop may execute many times**, depending on the number of iterations of the outer loop.

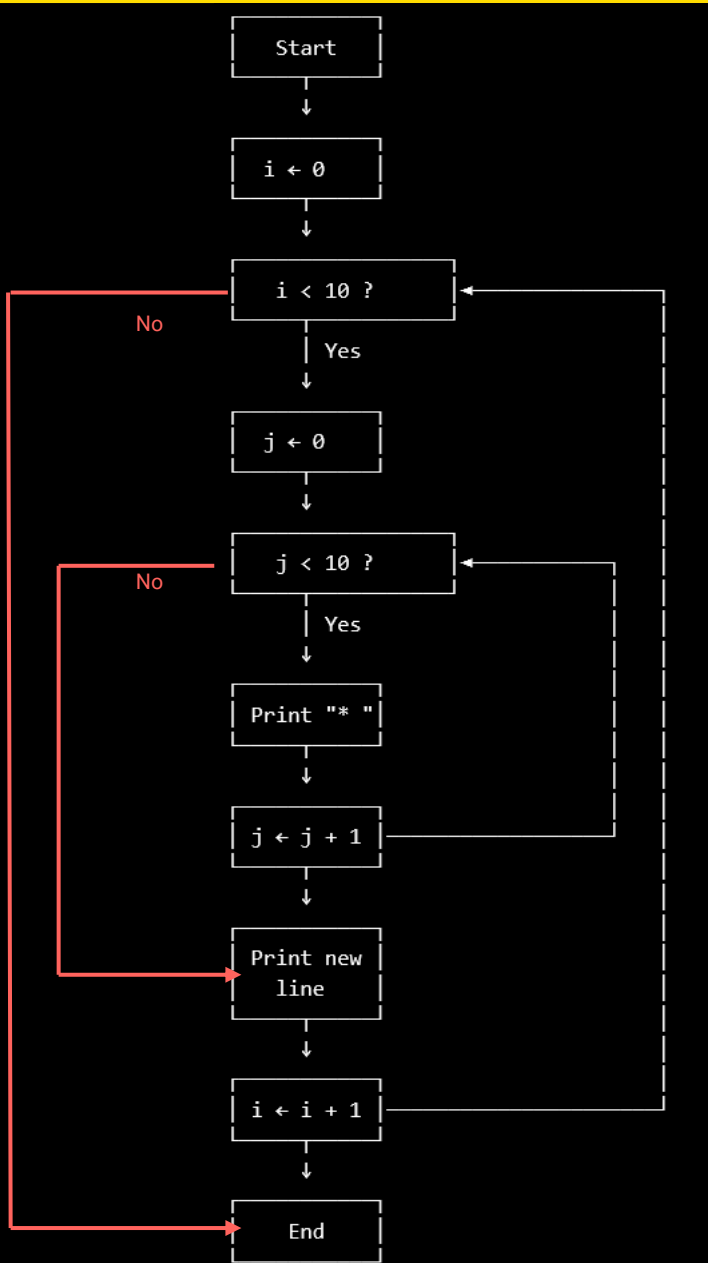
—→ Need a separate loop counter variable for each nested loop.

—→ For example, How can we produce a grid of * like this, using code?

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```



Example:



// print a square of 10x10 asterisks

```
int i = 0;  
while (i < 10) {
```

```
    int j = 0;  
    while (j < 10) {  
        printf("* ");  
        j = j + 1;  
    }
```

```
    printf("\n");  
    i = i + 1;  
}
```

What about a half-pyramid?

1
12
123
1234
12345

```
1 #include <stdio.h>
2
3 int main(void) {
4     int row = 1;
5
6     // Outer loop: for each row
7     while (row <= 5) {
8         int num = 1;
9
10        // Inner loop: print numbers from 1 up to current row number
11        while (num <= row) {
12            printf("%d", num);
13            num++;
14        }
15
16        // Move to next line after finishing a row
17        printf("\n");
18        row++;
19    }
20
21    return 0;
22 }
```

Demo

→ `while_recap.c`

→ `nested_loops.c`

Custom Data Types

- Up to this point, we've been using C's **built-in data types**, such as **int**, **char**, and **double**.
- Each of these types can hold **only a single value** at a time.
- But what if we need to **store a collection of related values together**?
- For example, information related to a **student**

```
#include <stdio.h>

int main(void) {
    int student_age = 21;
    char first_initial = 'A';
    int current_year = 3;

    return 0;
}
```

- The above variables are related ...
- We can define our own data types (structures) to store a collection of types.

structs

- structs hold multiple values (fields)
- struct are heterogeneous - fields can be different type
- struct field selected using name
- struct fields are fixed
- struct syntax:

```
struct <struct_name> {  
    data_type identifier;  
    data_type identifier;  
}
```

Example:

define a
struct that
holds a
student
details

```
struct student {  
    int zid;  
    double totallabMarks;  
    double assignment1Mark;  
    double assignment2Mark;  
}
```

To use, we simply say:

```
struct student pantea;
```

Notice, **no values**... we are only **defining**.

the . operator

To access the actual data of the structure:

```
// Declare a student variable  
struct student pantea;
```

```
// Assign values to the student's fields  
pantea.zid = 12345678;  
pantea.totallabMarks = 85.5;  
pantea.assignment1Mark = 40.0;  
pantea.assignment2Mark = 42.0;
```

The whole program:

```
1#include <stdio.h>
2
3// Define the student struct
4struct student {
5    int zid;
6    double totallabMarks;
7    double assignment1Mark;
8    double assignment2Mark;
9};
10
11int main(void) {
12
13    // Declare a student variable
14    struct student pantea;
15
16    // Assign values to the student's fields
17    pantea.zid = 12345678;
18    pantea.totallabMarks = 85.5;
19    pantea.assignment1Mark = 40.0;
20    pantea.assignment2Mark = 42.0;
21
22    // Print the student's details
23    printf("Student Details:\n");
24    printf("ZID: %d\n", pantea.zid);
25    printf("Total Lab Marks: %.2f\n", pantea.totallabMarks);
26    printf("Assignment 1 Mark: %.2f\n", pantea.assignment1Mark);
27    printf("Assignment 2 Mark: %.2f\n", pantea.assignment2Mark);
28
29    return 0;
30}
```

Demo

→ struct_book.c

Another custom data type:

enum

→ ENUMS (enumerations) is a custom data type, which describes set of possible values in a programmer-defined category

→ For example, days of the week

→ **Syntax**

```
enum enum_name { value1, value2, value3, ... };
```

→ **Example:**

```
enum weekdays { Mon, Tue, Wed, Thu, Fri, Sat, Sun };
```

Example:

```
1#include <stdio.h>
2
3// Define an enumeration for the days of the week
4enum weekdays { Mon, Tue, Wed, Thu, Fri, Sat, Sun };
5
6int main(void) {
7    // Declare a variable of type enum weekdays
8    enum weekdays day;
9
10   // Assign a value using one of the enumerated names
11   day = Sat;
12
13   printf("The value of day (Sat) is: %d\n", day);
14
15   return 0;
16}
```

- Enums let a variable hold only certain predefined values.
- They give easy-to-read names, like **Sat** for **Saturday**.
- No need to remember numbers for each value (0, 1, 2...).
- Using **#define** works, but too many can make the code messy

Struct and enum

```
// Define an enum for book condition
enum book_condition { New, Used, Damaged };

// Define a struct for a book
struct book {
    enum book_condition condition;
    double price;
};
```

Full program:

```
1#include <stdio.h>
2
3// Define an enum for book condition
4enum book_condition { New, Used, Damaged };
5
6// Define a struct for a book
7struct book {
8    enum book_condition condition;
9    double price;
10};
11
12int main(void) {
13    // Declare a book variable
14    struct book my_book;
15
16    // Assign values to the book
17    my_book.condition = Used;
18    my_book.price = 29.99;
19
20    // Print book details
21    printf("Book Details:\n");
22    printf("Price: $%.2f\n", my_book.price);
23
24    // Print condition as text
25    printf("Condition: ");
26    if (my_book.condition == New) {
27        printf("New\n");
28    } else if (my_book.condition == Used) {
29        printf("Used\n");
30    } else if (my_book.condition == Damaged) {
31        printf("Damaged\n");
32    }
33
34    return 0;
35}
```

Demo

→enum_coffee.c

→struct_enum_book.c

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 ...