DPST1091 / CPTG1391

Introduction to Programming

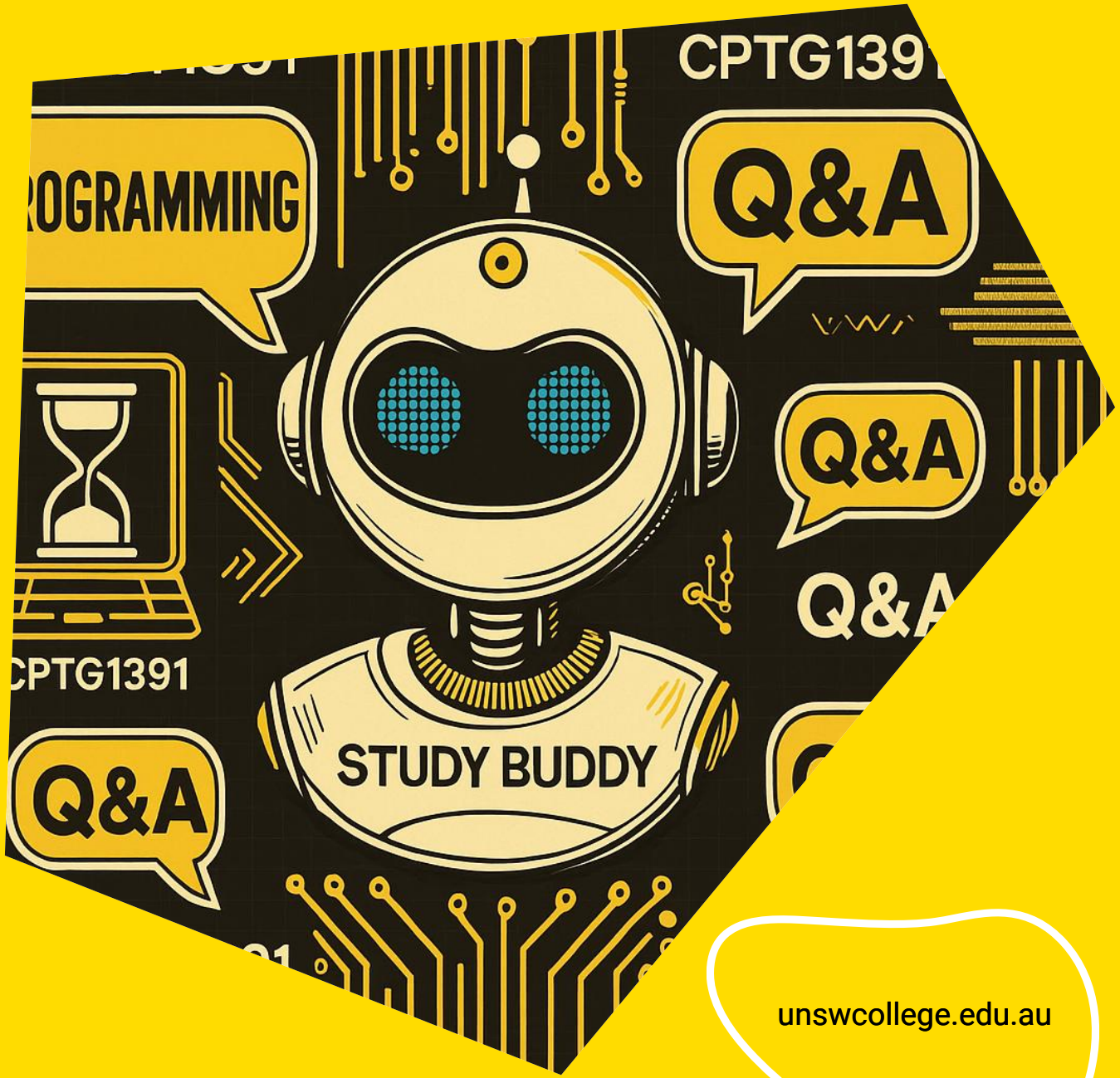
Week 2 – Lecture 2

Lecturer and Course Convener:

Dr Pantea Aria

UNSW College

Custom Data Types



unswcollege.edu.au

Agenda

- **Last lecture**
  - Control Flow
  - Conditions
  - Loops

- **Today**
  - Nested Loops
  - Custom Data Types

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;
}
```

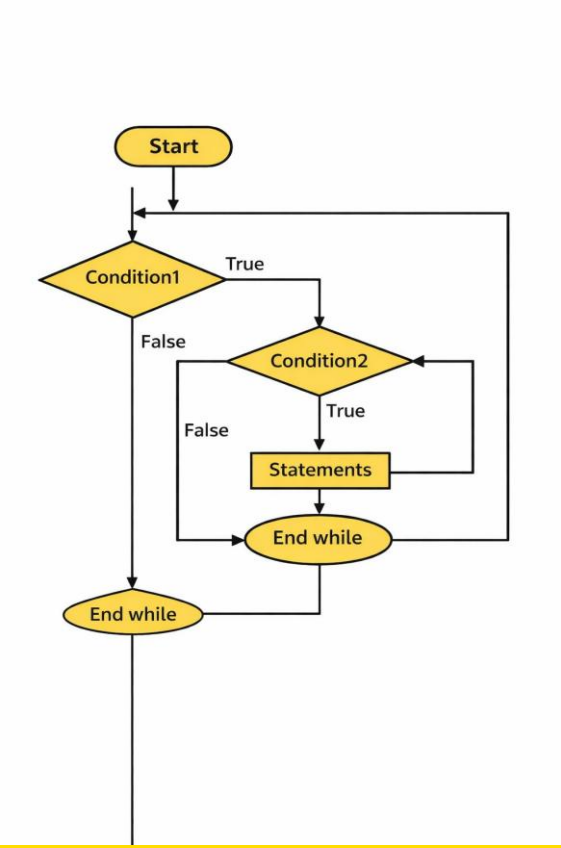
⟶If the condition is **true**, execute the body of the loop.

⟶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.

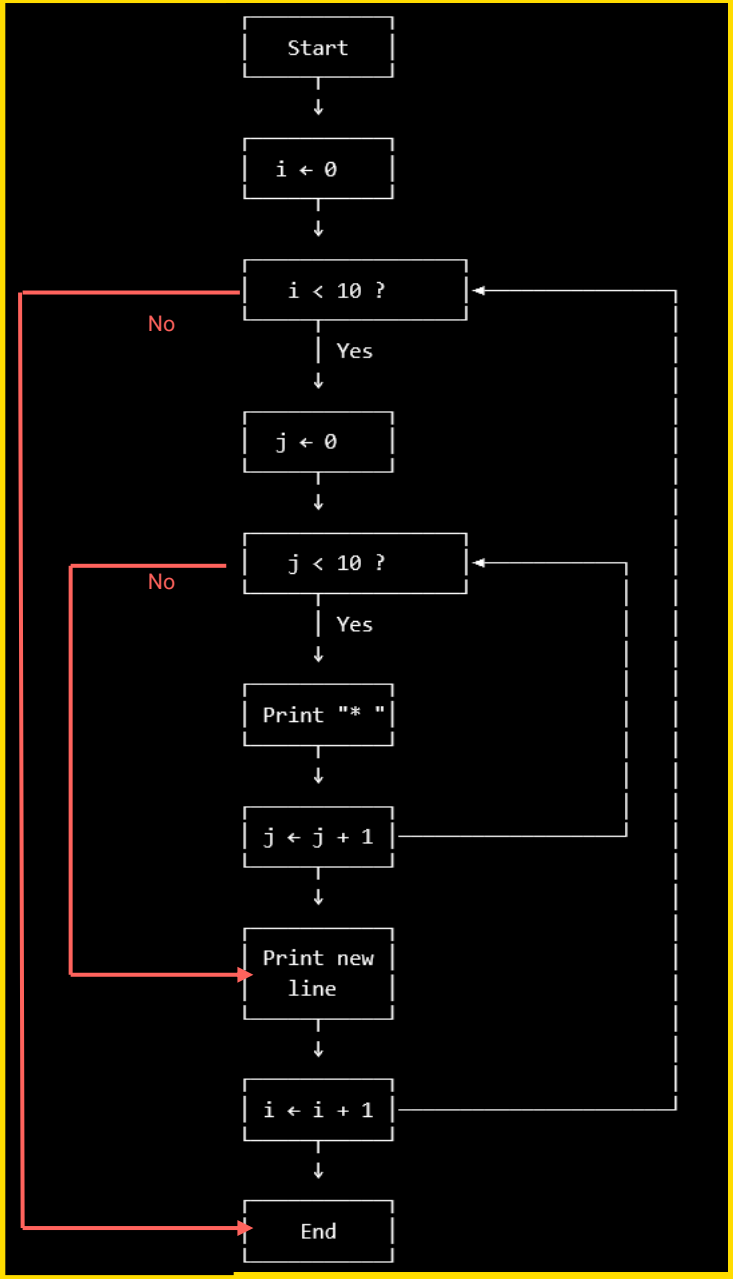
⟶This means the **inner loop may execute many times**, depending on the number of iterations of the outer loop.

⟶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
```

```c
#include <stdio.h>

int main(void) {
    int row = 1;

    // Outer loop: for each row
    while (row <= 5) {
        int num = 1;

        // Inner loop: print numbers from 1 up to current row number
        while (num <= row) {
            printf("%d", num);
            num++;
        }

        // Move to next line after finishing a row
        printf("\n");
        row++;
    }

    return 0;
}
```

Demo

→while_recap.c
→nested_loops.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

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**

```c
#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:

```c
#include <stdio.h>

// Define the student struct
struct student {
    int zid;
    double totallabMarks;
    double assignment1Mark;
    double assignment2Mark;
};

int main(void) {

    // 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;

    // Print the student's details
    printf("Student Details:\n");
    printf("ZID: %d\n", pantea.zid);
    printf("Total Lab Marks: %.2f\n", pantea.totallabMarks);
    printf("Assignment 1 Mark: %.2f\n", pantea.assignment1Mark);
    printf("Assignment 2 Mark: %.2f\n", pantea.assignment2Mark);

    return 0;
}
```

Demo

→struct_book.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!

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:

```c
#include <stdio.h>

// Define an enumeration for the days of the week
enum weekdays { Mon, Tue, Wed, Thu, Fri, Sat, Sun };

int main(void) {
    // Declare a variable of type enum weekdays
    enum weekdays day;

    // Assign a value using one of the enumerated names
    day = Sat;

    printf("The value of day (Sat) is: %d\n", day);

    return 0;
}
```

- 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

```c
// 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:

```c
#include <stdio.h>

// 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;
};

int main(void) {
    // Declare a book variable
    struct book my_book;

    // Assign values to the book
    my_book.condition = Used;
    my_book.price = 29.99;

    // Print book details
    printf("Book Details:\n");
    printf("Price: $%.2f\n", my_book.price);

    // Print condition as text
    printf("Condition: ");
    if (my_book.condition == New) {
        printf("New\n");
    } else if (my_book.condition == Used) {
        printf("Used\n");
    } else if (my_book.condition == Damaged) {
        printf("Damaged\n");
    }

    return 0;
}
```

Demo

⟶enum_coffee.c

⟶struct_enum_book.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