Féidearthachtaí as Cuimse
Infinite Possibilities

Semester 2
Week 2 - Tutorial

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Programming - Week 2 – 4th January 2025

Overview

- Functions – quick revision
- Returning a value from a function
- Return Types
- Pass By Value
- Pass By Reference
- Lab Mandatory Question

Function Signature

(aka Function Prototype) Revision

Function Name

Return type. This will be a specific data type or void

```
// Function declaration (prototype)
int add(int a, int b);
```

These are called a parameter. Parameter(s) are pieces of data that are passed to a function to use

Function Structure

Revision

- **Function Name**: Identifies the function.
  - It must follow naming rules (e.g., no spaces, cannot start with a digit).
- **Parameter List**: Specifies the inputs to the function (optional).
- **Function Body**: Contains the code to be executed.
- **Return Type**: Specifies the type of value the function returns.
  - If it doesn't return a value, use void.

Returning a value from a function

- **Functions can return data after they have completed**. This data can be a regular data type, e.g., an int, float, etc., or even a data structure such as an array.
- One important point to note is that a function can only return **a single data item**, i.e., it can only return a single int, float, etc., or a single data structure.

Function definition

Revision

Return type. This will be a specific data type or void

Function Name

These are called a parameter. Parameter(s) are pieces of data that are passed to a function to use

```
// Function definition
int add(int a, int b) {
    return a + b;  // Returns the sum of a and b
}
```

Specific return value

Return Types

- The **return type** of a function specifies what type of value the function will return to the caller.
- If a function does not return anything, it is declared as void.

```
// Function definition
int add(int a, int b) {
    return a + b;  // Returns the sum of a and b
}
```

Return Types Examples

Return Type	Description	Example Usage
void	No return value	void clear_screen();
int	Returns an integer	int add(int num1, int num2)
float	Returns a decimal number	float get_average(float a, float b)
double	Returns a decimal number	double inches_to_centimeters(double inches);
char	Returns a single character	char get_first_letter(char str[])
char*	Returns a string	char* get_greeting()
pointer	Returns a memory address	int* create_array(int size)
bool	Returns true or false	bool is_even(int num)
struct	Returns multiple values in a structure	struct Student get_student()

Example 1

- Create a program to calculate the area for the following shapes
  - Square
  - Triangle
  - Circle

Example 1 – code solution

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

C example_1.c ×

C example_1.c > main()

```c
#include <stdio.h>
#include <math.h> // to get access to Pi (M_PI)

double area_of_square(double side);
double area_of_triangle(double base, double height);
double area_of_circle(double radius);

```

- Math.h is giving access to Pi for the circle area calculation.
- The function declarations go before the main() function.

Example 1 – code solution

```c
int main() {
    double side, base, height, radius;
    double circle_area, triangle_area, square_area;

    // Input for square
    printf("Enter the side length of the square: ");
    scanf("%lf", &side);
    square_area = area_of_square(side);
    printf("Area of Square: %.2lf\n", square_area);

    // Input for triangle
    printf("Enter the base and height of the triangle: ");
    scanf("%lf %lf", &base, &height);
    triangle_area =  area_of_triangle(base, height);
    printf("Area of Triangle: %.2lf\n", triangle_area);

    // Input for circle
    printf("Enter the radius of the circle: ");
    scanf("%lf", &radius);
    circle_area = area_of_circle(radius);
    printf("Area of Circle: %.2lf\n", circle_area);

    return 0;
}
```

- This is the main function (the entry point for the program)
- We declare the variables for the shape sizes and to store the return values from the functions.
- The user is asked to enter the data for each shape, this is stored and passed to the functions to perform the area calculation.

Example 1 – code solution

```
33    // Function to calculate the area of a square
34    double area_of_square(double side) {
35        double area = side * side;
36        return area;
37    }
38
39    // Function to calculate the area of a triangle
40    double area_of_triangle(double base, double height) {
41        double area = 0.5 * base * height;
42        return area;
43    }
44
45    // Function to calculate the area of a circle
46    double area_of_circle(double radius) {
47        double area = M_PI * radius * radius;
48        return area;  // M_PI is a constant from math.h for π
49    }
50
```

- Implement the functions.
- Each function solves a specific logical problem.
- The function takes inputs which are required for the area calculation.
- The calculated area is returned to the caller.

Pass by Value vs. Pass by Reference

- In **C programming**, function arguments can be passed in two ways:
- **Pass by Value** – A copy of the variable is passed.
- **Pass by Reference** – A reference (memory address) is passed.

Pass By Value

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

C Example_2.c ●

C Example_2.c > ...

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

void to_uppercase(char answer);

int main() {
    char user_answer = 'y';
    printf("Char in main is initally: %c\n", user_answer);

    to_uppercase(user_answer);

    printf("Our char in main is now: %c\n", user_answer);

    return 0;
}

void to_uppercase(char answer) {
    answer = toupper(answer);  // Convert character to uppercase
    printf("Our char in to_uppercase is now: %c\n", answer);
}

```

- Passing a variable to a function.
- A **copy** of the variable value is **passed to the function**.
- Any changes made to this value stays within the function. ie. The **original value** in main **doesn't change.**

```
Char in main is initally: y
Our char in to_uppercase is now: Y
Our char in main is now: y
TU_Dublin >>
```

Pass By Reference

Example_3.c

```c
#include <stdio.h>

// Function to modify value using pointer
void its_your_birthday(int *num) {
    *num = *num + 1;  // Change will affect the original variable
    printf("Inside function: num = %d\n", *num);
}

int main() {
    int age = 18;
    printf("Before function call: num = %d\n", age);
    its_your_birthday(&age);  // Pass address of age
    printf("After function call: num = %d\n", age);
    return 0;
}
```

The function its_your_birthday(int *num) adds 1 to the age.
ie. The person is now 1 year older

- The function receives a **pointer (memory address)** of the variable.
- Changes **inside the function affect** the original variable.

```
Before function call: num = 18
Inside function: num = 19
After function call: num = 19
TU_Dublin >>
```

Summary

Feature	Pass by Value	Pass by Reference
What is passed?	A **copy** of the variable	A **memory address (pointer)**
Original value changes?	No	Yes
Used for?	Primitive data types (int, char, float)	Arrays, large structures, modifying values
Memory usage	More (creates a copy)	Less (works with original data)

Use Pass by Value when **we don't want** the function to modify the original variable (e.g., simple calculations).
Use Pass by Reference when **we need** the function to modify the original variable or for efficiency (e.g., modifying arrays, etc…).

Mandatory Question – in class solution

- Write a program that uses 2 functions called sum() and average().
- Your program must ask the user to enter 3 numbers inside the main().
- Your main() should then pass these 3 values as parameters to the function sum(). This function should calculate the sum of the 3 numbers.
- Your function sum() should then pass the sum of the 3 numbers as a parameter to the function average().
- The function average() should then calculate the average of the 3 numbers and display this on the screen.

Questions