Féidearthachtaí as Cuimse
Infinite Possibilities

Semester 2
Week 1 - Tutorial

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

Programming - Week 1 – 27th January 2025

Overview

- Functions
- What is a function
- Advantages of functions
- Using functions
- Code comparisons (using functions vs not)

What is a Function?

- A function in C is a self-contained block of code that performs a specific task.
- Functions allow us to break a program into smaller, more manageable, reusable components.
- It helps to make the program codebase easier to read, debug, and maintain.

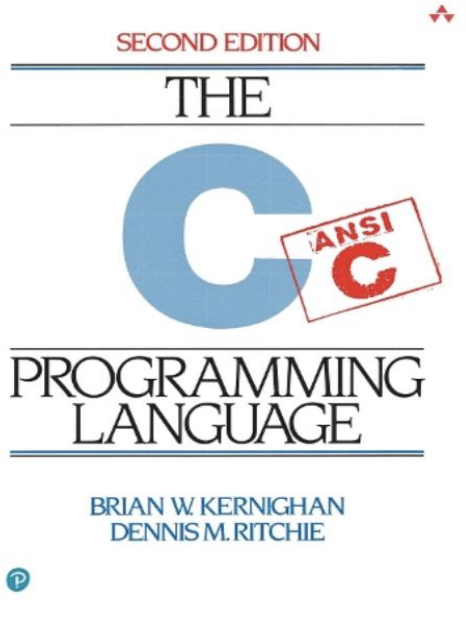
Types of Functions

- Library Functions:
  - Predefined functions provided by C standard libraries.
  - Examples: printf, scanf, etc…
- User-Defined Functions:
  - Functions created by the programmer for specific tasks.

Why use functions?

- A function provides a convenient way to encapsulate some computation, which can be used without worrying about its implementation.
- With a properly designed function we can ignore how a job is done, knowing what is done is sufficient.
- Eg. printf(“I am %d years old”, 18);

Question: What code is in the printf function?
Do we need to know this to use printf?



Source:
C Programming Language
Dennis Ritchie (author), Brian W. Kernighan (author)

Advantages of Using Functions

- Code Reusability: Write once, use multiple times.
- Modularity: Breaks down large programs into smaller parts.
- Readability: Easier to understand and maintain.
- Testing: Functions can be tested independently.

Function Structure

- **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.

Function Signature
(aka Function Prototype)

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 definition

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

Function Naming Conventions

- A function name must begin with an alphabetic letter or the underscore _ character, other characters in the name can be chosen from the followings:
  - Any lower-case letter from a to z
  - Any upper-case letter from A to Z
  - Any digit from 0 to 9
  - The underscore character _
- The function name should be a good representation of the functionality on offer.

Function – with no return

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

C Example_4.c > ...

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

// Function declaration (prototype)
void clear_screen();

int main() {
    printf("This text will be cleared.\n");
    clear_screen(); // function calls
    return 0;
}

// Function definition
void clear_screen() {
    system("clear"); // Use "cls" for Windows
    printf("Screen cleared.\n");
}
```

- The function declaration starts with **void** as there will be nothing returned from the function.
- No data is being passed to the clear_screen function. Ie. That's why nothing is listing in the ();

Function – int return

example_1.c

```c
#include <stdio.h>

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

// Main function
int main() {
    int x = 5, y = 10;
    int result = add(x, y);  // Function call
    printf("The sum is: %d\n", result);
    return 0;
}

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

- The function declaration starts with **int** as there will be an int value returned from the function.
- Data is being passed to the add function. Ie. That's why we have variables listing in the (int a, int b);
- Program to add two numbers together.
- The computational logic will be encapsulated in a function.

Example 2

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

C example_2.c > ...

```c
#include <stdio.h>

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

// Main function
int main() {
    int x, y;
    x = get_number_input();
    y = get_number_input();
    int result = add(x, y);  // Function call
    printf("The sum is: %d\n", result);
    return 0;
}

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

// Function definition
int get_number_input() {
    int number_input;
    printf("Enter a number: ");
    scanf("%d", &number_input);
    return number_input;  // Returns number input by user
}
```

The program extends the first example in the slides. The program will ask the user to input the numbers to be added together. The user input (scanf) has been added to a function that can be called whenever its needed.

Example 2 - extended

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TECHNOLOGICAL UNIVERSITY DUBLIN

C example_2.c > ...

```c
#include <stdio.h>

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

// Main function
int main() {
    int x, y;
    x = get_number_input(); // Function call
    y = get_number_input(); // Function call
    int result = add(x, y);  // Function call
    printf("The sum is: %d\n", result);
    return 0;
}
```

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

// Function definition
int get_number_input() {
    int number_input;

    while (1) { // Infinite loop, will break when valid input is given
        printf("Enter a number: ");
        if (scanf("%d", &number_input) == 1) {  // Valid integer input
            return number_input;  // Return the valid number
        } else {
            // Message for invalid inputss
            printf("Invalid input. Please enter a valid number.\n");
        }
    }
}
```

The program functionality has been extended to validate the users input to ensure they only enter a number. scanf("%d", &number_input) - returns 1 if the input matches the expected format (%d for integers), if invalid input is provided (like letters or symbols), scanf returns 0.

Comparison

- Let's have a look a a program using functions vs not using functions.

No Functions

Example_3.c > ...

```c
#include <stdio.h>

// Main function
int main() {
    int x, y, result;

    while (1) { // Infinite loop, will break when valid input is given
        printf("Enter a number: ");
        if (scanf("%d", &x) == 1) {  // Valid integer input
            break; // valid number - break out of while loop
        } else {
            // Message for invalid inputss
            printf("Invalid input. Please enter a valid number.\n");
        }
    }

    while (1) { // Infinite loop, will break when valid input is given
        printf("Enter a number: ");
        if (scanf("%d", &y) == 1) {  // Valid integer input
            break; // valid number - break out of while loop
        } else {
            // Message for invalid inputss
            printf("Invalid input. Please enter a valid number.\n");
        }
    }

    result = x + y;  // Function call
    printf("The sum is: %d\n", result);
    return 0;
}
```

- Code logic has been repeated in the user inputs (while loops)
- Readability of the code suffers, what does this program do? Not obvious without going through the full codebase.
- Changes to the input code must be changed in multiple places (prone to human error)

Using Functions

C example_2.c > ...

```c
#include <stdio.h>

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

// Main function
int main() {
    int x, y;
    x = get_number_input(); // Function call
    y = get_number_input(); // Function call
    int result = add(x, y);  // Function call
    printf("The sum is: %d\n", result);
    return 0;
}

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

// Function definition
int get_number_input() {
    int number_input;

    while (1) { // Infinite loop, will break when valid input is given
        printf("Enter a number: ");
        if (scanf("%d", &number_input) == 1) {  // Valid integer input
            return number_input;  // Return the valid number
        } else {
            // Message for invalid inputss
            printf("Invalid input. Please enter a valid number.\n");
        }
    }
}
```

- Good separation of concerns
- Following the DRY principle
  - **D**on't **R**epeat **Y**ourself
- Easy to understand what the program does, main codebase is smaller and easy to read / understand.

Programming Pitfall

- We need to declare the functions before main() in our codebase.
- If a function is returning data it must set the return type:
  - Eg. **int** get_number_input() { …
  - And **return** the data
- If no data is returned the return type is **void**
  - void clear_screen() {
  - With **no return** in the function

Questions