計算機程式設計

Computer Programming

Functions

Instructor: 林英嘉

2024/10/21

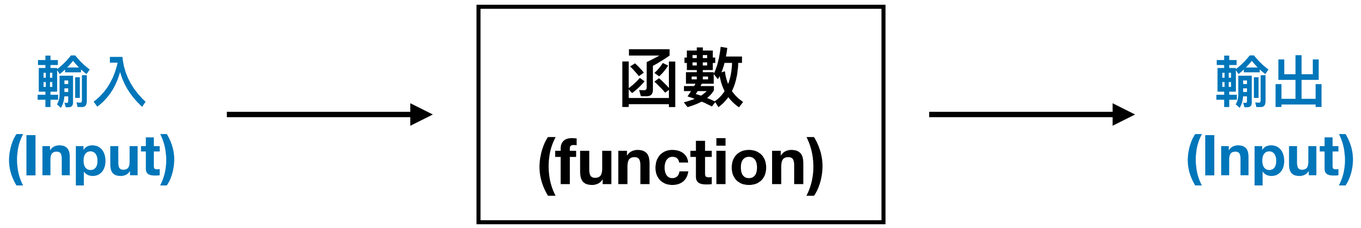
W7 Slido: #3533560

GitHub repo

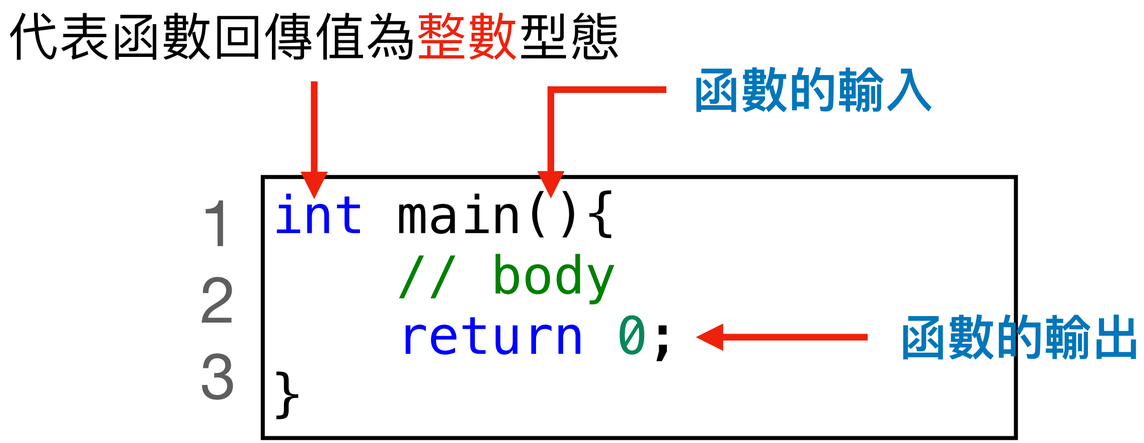
Outline

- Definition of “functions”
- Parameters and Arguments
- Practical examples
  - How many days in a month? (with leap_year)
  - Prime number counter
  - Use return in functions

[Recap] 函數



[Recap] C語言程式骨架



- return 0; 代表main函數執行完畢，回傳整數0
  - 多數時候，main函數中沒有寫這行不會有問題
- 程式是一行一行執行，也就是body執行完後才會執行return 0;那行
- 注意分號

Two Categories of Functions

- **System** functions
  - Pre-defined functions provided by the C standard library
- **Custom** functions
  - Functions you define (you write) to perform specific tasks

[Recap] stdio.h (standard input / output) 有什麼？

(僅列出部分函式)

Formatted input/output:

fprintf	Write formatted data to stream (function)
fscanf	Read formatted data from stream (function)
printf	Print formatted data to stdout (function)
scanf	Read formatted data from stdin (function)
snprintf	Write formatted output to sized buffer (function)
sprintf	Write formatted data to string (function)
sscanf	Read formatted data from string (function)
vfprintf	Write formatted data from variable argument list to stream (function)
vfscanf	Read formatted data from stream into variable argument list (function)
vprintf	Print formatted data from variable argument list to stdout (function)
vscanf	Read formatted data into variable argument list (function)
vsnprintf	Write formatted data from variable argument list to sized buffer (function)
vsprintf	Write formatted data from variable argument list to string (function)
vsscanf	Read formatted data from string into variable argument list (function)

Figure source: https://cplusplus.com/reference/cstdio/

System functions

- How to know what the C standard library includes? https://cplusplus.com/reference/
- Six common header files:

header file	Purpose	Examples
stdio.h	Standard I/O	printf(), scanf(), …
stdlib.h	Standard library (general tools)	abs(), malloc(), free(), exit(), …
math.h	math	sqrt(), pow(), sin(), cos(), ….
ctype.h	Functions related to characters	isalpha(), isdigit(), toupper(), …
string.h	Functions for processing strings	strcpy(), strlen(), strcmp(), …
time.h	time	time(), clock(), difftime(), …

Each header has its own topic!

Two Categories of Functions

- **System** functions
  - Pre-defined functions provided by the C standard library
- **Custom** functions
  - Functions you define (you write) to perform specific tasks

How to write a custom function in your program?

```
func_prototype; // 函數原型

main(){
  main_body;
}

func(){
  func_body;
}
```

main function

custom function

- [Reminder] A program is compiled from the top to the bottom.

Function Prototype (函數原型)

- A function prototype is:

```
return_type func_name(type1, type2, ...);
```

- Purposes:
  1. **Type Checking:** Help the compiler **check the correctness of data types** when you use a function in the main function.
  2. **Function Declaration**: Allows function calls before the function is defined.

- You can also write a function prototype as the following to increase readability:

```
return_type func_name(type1 param1, type2 param2, ...);
```

Declaration of Functions

```
return_type func_name(type1 param1, type2 param2, ...){
    body;
    return value;
}
```

	return_type	func_name	param_type	param_name	body	return statement
prototype	✓	✓	✓	optional	✗	✗
declaration	✓	✓	✓	✓	✓	✓

- You need to add **a semicolon** at the end of **each line in body** and **the return statement.**

Simple Example

C-course-materials/05-Functions/basic.c

```c
#include <stdio.h>
int add(int, int); // prototype

int main() {
    int result = add(3, 4);
    printf("Result: %d", result);
    return 0;
}

int add(int a, int b) {
    return a + b;
}
```

Parameters and Arguments

- Parameters (參數); formal parameters
- Arguments (引數); actual parameters
- A **parameter** is a variable in a **method definition**. When a method is called, the **arguments** are **the data you pass** into the method's parameters.

```
return_type func_name(type1 param1, type2 param2, …){
   body;
}
...

func_name(myArg1, myArg2);
```

Let’s see the reference of printf

- https://cplusplus.com/reference/cstdio/printf/

Expressions can be used as arguments

C-course-materials/05-Functions/input_expression.c

- An argument does not need to be a value.

```c
#include <stdio.h>
int add(int, int); // prototype

int main() {
    int result = add(3 + 3, 4 / 2);
    printf("Result: %d", result);
    return 0;
}

int add(int a, int b) {
    return a + b;
}
```

Prototype -> main -> func

main function

custom function

```
func_prototype; // 函數原型

main(){
  main_body;
}

func(){
  func_body;
}
```

- [Reminder] A program is compiled from the top to the bottom.

func -> main

custom function

main function

```
func(){
  func_body;
}

main(){
  main_body;
}
```

- [Reminder] A program is compiled from the top to the bottom.

Complete main() Program

C-course-materials/05-Functions/complete_main.c

```c
#include <stdio.h>
int main(void){
    printf("This is week 7.");
    return 0;
}
```

- **void** is a type with no value.
- Generally, main function does not require a parameter. Placing **void** inside a main function adheres to C Standards and increases clarity.

Place void inside a custom function

C-course-materials/05-Functions/custom_void.c

```c
#include <stdio.h>
int add(void) {
    printf("This is week 7.\n");
    return 0;
}
int main(void){
    int result = add();
    printf("This is week 7.\n");
    printf("Result: %d", result);
    return 0;
}
```

- **void** is a type with no value.
- For custom functions that do not require a parameter, we should also add **void** for clarity and readability, though it will not raise compilation errors if **void** is not placed.

Use return to end earlier

- We can use the return statement to end a function early.
- This technique can be used in many cases, such as:
    1. **Error check**
    2. **End a loop earlier**

Error check: How many days in a month?

Write a program that accepts a positive integer n as **a month**, and the program must output the number of days for that month. Assume this is all in a leap year.

Input

```
12
```

Output

```
Number of days in month 12 is 31.
```

Input

```
13
```

Output

```
Invalid month number.
```

How many days in a month? (function)

C-course-materials/05-Functions/num_days_in_a_month.c

```c
#include <stdio.h>
int get_days(int month, int leap_year){
    if (month < 1 || month > 12)
        return -1;
    else{
        if (month == 2){
            if (leap_year == 1)
                return 29;
            else
                return 28;
        }
        else if (month == 4 || month == 6 || month == 9 || month == 11)
            return 30;
        else
            return 31;
    }
}
```

How many days in a month? (main)

C-course-materials/05-Functions/num_days_in_a_month.c

```c
int main(){
    int month, leap_year = 1;
    scanf("%d", &month);
    int days = get_days(month, leap_year);
    if (days == -1)
        printf("Invalid month number.");
    else
        printf("Number of days in month %d is %d.", month, days);
    return 0;
}
```

Leap Year

- There are 365.24219 days in a year.
- An extra calendar day is added every four years.
  - However, there will be big difference between 0.242 and 0.25 for many years.
- In the Gregorian calendar now in general use, the discrepancy is adjusted by adding the extra day to **only those century years exactly divisible by 400** (e.g., 1600, 2000).

How many days in a month? (function and main)

C-course-materials/05-Functions/num_days_in_a_month_complete.c

```c
int is_leap_year(int year){
    int is_leap = ((year%400==0)||(year%4==0 && year%100!=0))? 1: 0;
    return is_leap;
}
```

```c
int main(){
    int month, leap_year;
    int year = 2024;
    scanf("%d", &month);
    leap_year = is_leap_year(year);
    printf("%d is a leap year: %d\n", year, leap_year);
    int days = get_days(month, leap_year);
    if (days == -1)
        printf("Invalid month number.");
    else
        printf("Number of days in month %d is %d.", month, days);
    return 0;
}
```

Use return to end earlier

- We can use the return statement to end a function early.
- This technique can be used in many cases, such as:

1. **Error check**
2. **End a loop earlier**

End a loop earlier: Prime Number Counter

Write a program that accepts a positive integer n, and the program must output the number of prime numbers between 1 and n (inclusive). An example is shown below.

Input

```
10
```

Output

```
4
```

Prime Number Counter (function)

C-course-materials/05-Functions/prime_numbers.c

```c
#include <stdio.h>

// return 1 if num is prime, otherwise return 0
int is_prime(int num) {
    if (num < 2) {
        return 0;
    }
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}
```

Prime Number Counter (main)

C-course-materials/05-Functions/prime_numbers.c

```c
int main(void) {
    int n, primeCount = 0;
    printf("請輸入一個正整數 n: ");
    scanf("%d", &n);
    for (int i = 2; i <= n; i++) { // start from 2
        if (is_prime(i)) {
            primeCount++;
        }
    }
    printf("%d", primeCount);
}
```

Use return; for a function without returning a value

- Write a program that takes a number as input. The program will read the input to store each digit into an array and finally print the array.

- Input

```
12345
```

- Output

```
1 2 3 4 5
```

Use return; for a function without returning a value

C-course-materials/05-Functions/return_no_value.c

```c
#include <stdio.h>
void print_array(int arr[], int size){
    for (int i=0; i<size; i++){
        if (i == 0)
            printf("%d", arr[i]);
        else
            printf(" %d", arr[i]);
    }
    return;
}
int main(void){
    int arr[5];
    int i = 0;
    while (i < 5){
        scanf("%1d", &arr[i]);
        i++;
    }
    print_array(arr, 5);
    return 0;
}
```

- You should use **void** as the return_type if the function does not return anything.

Addresses between Parameters and Arguments

- Parameters (參數); formal parameters
- Arguments (引數); actual parameters
- A **parameter** is a variable in a **method definition**. When a method is called, the **arguments** are **the data you pass** into the method's parameters.

```
return_type func_name(type1 param1, type2 param2, …){
   body;
}
...

func_name(myArg1, myArg2);
```

Addresses between Parameters and Arguments

C-course-materials/05-Functions/param_arg_addr.c

```c
#include <stdio.h>

void increment_and_print(int func_num){
    printf("func_num address before ++: %p\n", &func_num);
    func_num++;
    printf("func_num address after ++: %p\n", &func_num);
    return;
}
int main(void){
    int input_num = 10;
    printf("input_num address before calling: %p\n", &input_num);
    increment_and_print(input_num);
    printf("input_num address after calling: %p\n", &input_num);
    return 0;
}
```

- Generally, parameters and arguments have **different memory addresses**.

Input Array to a Function

prototype

```
return_type func_name(type arr[], type2 param2, ...)
```

main function

```
int main(void){
  array_declaration;
  ...
}
```

custom function

```
return_type func_name(type arr[], type2 param2, ...){
   body;
   return value;
}
```

Not required

Not required

Addresses between Parameters and Arguments

C-course-materials/05-Functions/array.c

```c
#include <stdio.h>

void modifyArray(int arr[]) {
    printf("Address of parameter 'arr' inside function: %p\n", arr);
    arr[0] = 999; // Modify the first element
}

int main(void) {
    int score[4] = {80, 85, 90, 100};
    printf("Address of argument 'score' in main: %p\n", score);

    printf("First element before modification: %d\n", score[0]);
    modifyArray(score);
    printf("First element after modification: %d\n", score[0]);
    return 0;
}
```