變數的初始化

```
int age = 21; // C-like initialization
int age (21); // Constructor initialization
int age {21}; // C++11 list initialization syntax
```

🔷 第一種：C-like 初始化

`int age = 21;`

✅ 說明：

- 最常見、最傳統的寫法。
- **C 語言風格的賦值初始化**。
- 在函式內、全域、甚至 class 中都能用。
- 初學者最容易理解。

⚠️ 注意：

- 這種寫法在某些情況下會進行 **隱式轉型（implicit narrowing conversions）**，可能藏有錯誤。

```
int x = 3.14; // ✅ 編譯成功，但小數部分被截斷成 3
```

🔷 第二種：Constructor（函式風格）初始化

`int age(21);`

✅ 說明：

- 看起來像呼叫建構子，但對於基本型別（像 `int`），編譯器把它視為初始化。
- 這是 C++ 引入的語法風格，也適用於物件：

```
std::string s("Hello");
```

⚠️ 注意：

- 可能與函式宣告搞混。
  例如：

```
int age(); // ⚠ 這不是變數，是函式宣告！會讓人誤解
```

🔷 第三種：C++11 列表初始化（Uniform Initialization）

```
int age {21};
```

✅ 說明：

- C++11 引入的新語法，稱為 **brace initialization** 或 **uniform initialization**。
- 避免隱式轉型錯誤（narrowing conversion）。
- 支援 `std::initializer_list` 物件的初始化。
- 更一致的初始化風格（uniform）。

🛡️ 安全性範例：

```
int x {3.14}; // ❌ 錯誤：不能把 double 隱式轉成 int
int y = 3.14; // ✅ 可編譯，但會被截斷成 3
```

🧠 什麼是隱式轉型？

當你在程式中混用了不同資料型別，**編譯器會自動幫你轉型成合適的型別**，這種自動轉換就叫做 **隱式轉型（implicit conversion）**。

你不需要手動寫轉型的程式碼，C++ 會自己幫你處理。

🔁 範例一：基本數值型別

```
int i = 10;
double d = i; // int 被「自動轉型」成 double
```

這裡 `i` 是 `int`，而 `d` 是 `double`，C++ 會自動幫你轉成 `10.0`。

🔁 範例二：混合運算

```
int a = 3;
float b = 4.5;
auto result = a + b; // a 自動轉成 float，result 也是 float
```

這裡 `a` 是 int，`b` 是 float。因為兩者相加，**C++ 會把「較小的型別轉為較大的型別」**（這是「型別提升」的一部分），所以 `a` 會轉成 `float`。

📦 更進一步：物件與建構子的隱式轉型

如果你的類別定義了「只有一個參數的建構子（constructor）」，那它就可以被用來做隱式轉型。

✨ 範例：

```
class MyInt {
public:
  int value;
  MyInt(int v) : value(v) {} // 可用於隱式轉型
};

void print(MyInt m) {
  std::cout << m.value << "\n";
}

int main() {
  print(5);  // 這裡 5 是 int，但會被「隱式轉型」成 MyInt(5)
}
```

🚫 怎麼禁止這種隱式轉型？

有時你不希望發生隱式轉型，這時可以在建構子前加上 `explicit` 關鍵字：

```
class MyInt {
public:
  int value;
  explicit MyInt(int v) : value(v) {} // 禁止隱式轉型
};
```

C++ Primitive Data Types

Character Types

Type Name	Size/Precision
char	Exactly one byte. At least 8 bit
char16_t	At least 16 bits
char32_t	At least 32 bits
wchar_t	Can represent the largest available character set. cter

Integer Types

Type Name	Size/Precision
signed short int	At least 16 bits
signed int	At least 16 bits
signed long int	At least 32 bits
signed long long int	At least 64 bits

Type Name	Size/Precision
unsigned short int	At least 16 bits
unsigned int	At least 16 bits
unsigned long int	At least 32 bits
unsigned long long int	At least 64 bits

Floating-point Type

- Used to represent non-integer numbers
- Represented by mantissa and exponent (scientific notation)
- Precision is the number of digits in the mantissa
- Precision and size are compiler dependent

Type Name Size	/ Typical Precision	Typical Range
float	/7 decimal digits	1.2 * 10 ^ - 38 to 3.4 * 10 ^ 38

Type Name Size	/ Typical Precision	Typical Range
double	No less than float / 15 decimal digits	2.2 x 10-308 to 1.8 * 10 ^ 308
long double	No less than double / 19 decimal digits	3.3 * 10 ^ - 4932 to 1.2 * 10 ^ 4932

Boolean Type

- Used to represent true and false
- Zero is false.
- **Non-zero** is true.

Type Name	Size/Precision
bool	Usually 8 bits true or false (C++ keywords)

使用 sizeof operator

- sizeof operator用來鑑定一種type或一個variable的大小(bytes)
- Examples:

```
sizeof(int);
sizeof(double);
sizeof(some_variable);
sizeof some_variable;
```

<climits> 和 <cfloats>

<clmits>

✅ 說明：

提供**整數型別的限制常數**，例如 `int`、`long`、`char` 等型別的最大與最小值。

✅ 常見的巨集：

常數	說明
`CHAR_BIT`	一個 `char` 佔幾個 bit（通常是 8）
`CHAR_MIN` / `CHAR_MAX`	`char` 型別的最小 / 最大值
`INT_MIN` / `INT_MAX`	`int` 型別的最小 / 最大值
`LONG_MIN` / `LONG_MAX`	`long` 型別的最小 / 最大值
`USHRT_MAX` / `UINT_MAX` / `ULONG_MAX`	各 unsigned 型別的最大值（無負…

`<cfloat>`

✅ 說明：

提供**浮點數型別的限制常數**，如 `float`、`double`、`long double` 的最大值、最小值、有效位數等。

✅ 常見的巨集：

常數	說明
`FLT_MIN` / `FLT_MAX`	`float` 型別的最小 / 最大值
`DBL_MIN` / `DBL_MAX`	`double` 型別的最小 / 最大值
`LDBL_MIN` / `LDBL_MAX`	`long double` 的最小 / 最大值
`FLT_DIG` / `DBL_DIG`	小數點後可保證的有效位數

Constant

Constant的主要特色就是一經宣告就不能被修改!

Types of constants in C++

- Literal constants
- Declared constants
  - const keyword
- Constant expressions
  - constexpr keyword
- Enumerated constants

    - enum keyword
- Defined constants
    - #define

🟦 1. Literal constants（字面常數）

這是最基本的常數，直接寫在程式中的數字、字元、字串等就是字面常數。

```
#include <iostream>
int main() {
   int a = 10;         // 10 是一個整數常數（integer literal）
   char c = 'A';       // 'A' 是一個字元常數（character literal）
   double pi = 3.14;    // 3.14 是一個浮點常數（floating-point literal）
    std::cout << "a = " << a << ", pi = " << pi << ", c = " << c << std::endl;    return 0; }`
```

- literal constants的後綴
  12 - an integer
  12U - an unsigned integer
  12L - a long integer
  12LL - a long long integer
- Character Literal Constants (escape codes)
  \n -new line
  \r -return
  \t -tab
  \b -backspace
  \' -single quote
  \" -double quote
  \\ -backslash

🟦 2. Declared constants（宣告式常數）

✅ 使用 `const` 關鍵字，不能被修改。

```
#include <iostream>
int main() {
   const int DAYS_IN_WEEK = 7;    //
   DAYS_IN_WEEK = 8; // ❌ 錯誤：不能改變 const 常數
   std::cout << "A week has " << DAYS_IN_WEEK << " days." << std::endl;
   return 0;
}
```

🟦 3. Constant expressions（常數運算式）

✅ 使用 `constexpr`：在 編譯期就能確定值的常數。

```
#include <iostream>
constexpr int getSquare(int x) {
    return x * x;
}
int main() {
    constexpr int result = getSquare(5);
    // 編譯期就能計算出 result = 25
    std::cout << "Square of 5 is " << result << std::endl;
    return 0;
}
```

📌 `constexpr` 通常搭配 **inline 計算 或陣列大小、模板參數等需編譯期常數的地方**使用。

🟦 4. Enumerated constants（列舉常數）

✅ 使用 `enum` 關鍵字：一組命名的整數常數。

```
#include <iostream>
enum Color { RED, GREEN, BLUE };
int main() {
    Color c = GREEN;
    std::cout << "The value of GREEN is " << c << std::endl; // 輸出為 1（從 0 開始編號）
     return 0;
}
```

📌 可以指定值：

```
enum StatusCode { OK = 200, NOT_FOUND = 404, SERVER_ERROR = 500 };
```

🟦 5. Defined constants（定義式常數）

✅ 使用 `#define` 巨集預處理指令。

```cpp
#include <iostream>
#define PI 3.14159
#define SQUARE(x) ((x) * (x)) // 巨集函數
int main() {
   double area = PI * SQUARE(5);
   std::cout << "Area of circle with r=5 is " << area << std::endl;
   return 0;
   }
```

📌 缺點：沒有型別檢查，容易出錯，例如：

```cpp
SQUARE(x+1)  // 展開為 ((x+1) * (x+1)) → 正確
SQUARE x+1   // 展開為 ((x+1) * x+1) → 錯誤！（沒有加括號）
```

◆ 總結對照表：

類型	用途與特性
Literal constants	程式中直接出現的值，如 `10` 、 `'A'` 、 `3.14`
`const`	宣告不可改變的變數，具型別檢查
`constexpr`	編譯期常數，可用於陣列大小、模板等
`enum`	一組具名的整數常數，常用於狀態、選項表示
`#define`	預處理階段取代，無型別檢查，適合簡單常數與巨集