

Section 11 String

C-style Strings

Sequence of characters

- contiguous in memory
- implemented as an array of characters
- terminated by a null character (null)
- null - character with a value of zero
- Referred to as zero or null terminated strings

String literal

- sequence of characters in double quotes, e.g. "Frank"
- constant
- terminated with null character

C-Style string 是什麼？


C-Style string 就是一串以 null 字元 (\0) 結尾的字元陣列。
這個 null 字元是用來表示字串結束的位置。

◆ 範例:

```
char greeting[] = "Hello";
```

其實它等價於：

```
char greeting[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

 宣告方式

1. 使用字串常數（自動加上 \0）：

```
char str[] = "Apple";
```

2. 手動輸入字元（必須自己加 \0）：

```
char str[] = {'A', 'p', 'p', 'l', 'e', '\0'};
```

3. 使用指標（常搭配不可修改字串）：

```
char *str = "Banana";
```

//⚠ 這樣的寫法在某些情況下（如寫入）可能會導致錯誤，因為 "Banana" 儲存在唯讀區。

常用操作函數（來自 <string.h>）

函數	功能描述
strlen()	回傳字串長度（不含 \0）
strcpy()	複製字串
strcat()	字串接在另一個字串後面
strcmp()	比較兩字串（回傳整數）
strchr()	找到字元第一次出現的位置
strstr()	找子字串

範例：

```
#include <stdio.h>
#include <string.h>
int main() {
    char str1[20] = "Hello";
    char str2[] = "World";
    strcat(str1, str2); // str1 現在變成 "HelloWorld"
    printf("%s\n", str1);    return 0; }
```

注意事項



1. C-Style string 需要小心處理記憶體空間大小，避免 **buffer overflow**。
2. 字串操作函數不會自動檢查空間是否足夠。
3. 每個字串都 必須以 \0 結尾，否則會造成錯誤行為（如無限輸出）。

std::string 是什麼？

std::string 是 C++ 標準函式庫裡的一個類別 (class)，定義在 <string> 標頭檔中。它內部幫你管理記憶體、提供運算子重載和許多有用的字串操作函數。

```
#include <string> // 必須包含這個！
std::string str = "Hello, world!";
```

優點比較（與 C-style string 相比）

項目	C-style string (char[])	C++ string (std::string)
記憶體管理	手動處理	自動處理（建構子、解構子）
安全性	容易越界、未定義行為	有邊界檢查、支援例外處理
操作介面	需用 strcpy() 、 strlen() 等	可用 + 、 .size() 、 .substr() 等方法
動態長度	需要動態配置	自動擴充
可以用 == 比較嗎？	 不行（只能比指標）	 可以直接比較內容

基本用法

1. 宣告與初始化

```
#include <iostream>
#include <string>

int main() {
    std::string str1 = "Hello";
    std::string str2("World");
    std::string str3; // 空字串

    std::cout << str1 << " " << str2 << std::endl;
    return 0;
}
```

2. 串接字串 (+)

```
std::string name = "Alice";
std::string greet = "Hello, " + name + "!";
```

3. 比較字串 (==, <, >, ...)

```
if (name == "Alice") {
    std::cout << "Hi Alice!" << std::endl;
}
```

3. 長度 .length() 或 .size()

```
std::cout << name.length(); // 回傳字元數 (不含 \0)
```

3. 存取單一字元 (像陣列一樣)

```
char first = name[0]; // 'A'
```

3. 子字串 .substr()

```
std::string s = "banana";
std::string sub = s.substr(1, 3); // 從 index 1 開始取 3 個字元 → "ana"
```

3. 尋找子字串 .find()

```
std::string s = "banana";
size_t pos = s.find("na"); // 回傳 2
```

3. 轉回 C-style string (例如要給 C 函式用)

```
const char* cstr = str.c_str();
```

C++ std::string 建構子的各種方式

✅ 1. 從 C-style 字串建構 (最常見)

```
const char* cstr = "Hello world"; std::string str(cstr); // ← 直接用 C-style 字串初始化
```

或更常見的縮寫寫法：

```
std::string str = "Hello world"; // 編譯器會自動用 const char* 建構
```

✓ 2. 從另一個 `std::string` 建構（拷貝建構子）

```
std::string s1 = "Hi"; std::string s2(s1); // 複製 s1 的內容到 s2
```

✓ 3. 從某段 C-style 字串建構（只取前 N 個字元）

```
const char* msg = "HelloWorld";  
std::string part(msg, 5); // 只取前 5 個字元 → "Hello"
```

✓ 4. 重複某個字元建構字串

```
std::string s(10, '*'); // s = "*****" (10 個星號)
```

✓ 5. 從字串中的一段建構新字串

```
std::string s1 = "Hello world"; std::string s2(s1, 6); // s2 = "world" (從 index 6 開始到結尾)  
std::string s3(s1, 0, 5); // s3 = "Hello" (從 index 0 取 5 個) ``
```

✓ 6. 從初始化列表建構（C++11 以上）

```
std::string s({'H', 'i', '!'}); // s = "Hi!"
```

小補充：為什麼可以寫 `std::string str = "abc";` ？

這其實是因為有一個這樣的建構子：

```
string(const char* s);
```

也就是說：

```
std::string str = "abc"; // 是這樣的語法糖： std::string str("abc");`
```

而這是 隱式建構 (implicit constructor) 的應用。

範例整合

```
#include <iostream>
#include <string>
int main() {
    std::string a = "Hello"; // 從 C-style string
    std::string b(a); // 拷貝建構
    std::string c(a, 2); // 從 a 的第 2 個字元開始 → "llo"
    std::string d("World", 3); // 從 C-style 取前 3 個字元 → "Wor"
    std::string e(5, '!'); // 5 個驚嘆號 → "!!!!!"
    std::cout << a << " " << b << " " << c << " " << d << " " << e << std::endl;
    return 0;
}
```

總結表格

建構方式	範例	說明
從 C-style string	<code>std::string s = "hi";</code>	常見且簡潔
拷貝建構子	<code>std::string s2(s1);</code>	複製另一個 string
取部分 C-style string	<code>std::string s("hello", 3);</code>	得到 "hel"
從部分 string 建構	<code>std::string s(s1, 3);</code>	從 index 3 起取
取特定位數 string	<code>std::string s(s1, 2, 4);</code>	從 index 2 起取 4 個字元
重複某字元	<code>std::string s(10, '#');</code>	"#####"

記憶體與效能

`std::string` 通常是動態配置的 (heap)，但實作上可能用 SSO (Small String Optimization) 來避免小字串開堆疊記憶體。

你不需要手動釋放空間，C++ 會幫你處理。

 進一步範例：比較 C string vs C++ string

```
#include <iostream>
#include <cstring>
#include <string>

int main() {
    // C-style
    char cstr1[100] = "Hello";
    char cstr2[] = "World";
    strcat(cstr1, " ");
    strcat(cstr1, cstr2);
    std::cout << "C-style: " << cstr1 << std::endl;

    // C++ string
    std::string str1 = "Hello";
    std::string str2 = "World";
    std::string result = str1 + " " + str2;
    std::cout << "C++ string: " << result << std::endl;

    return 0;
}
```

🚩 總結

功能	std::string 提供的方式
建構字串	std::string s = "abc";
串接字串	s1 + s2
比較字串	==, !=, <, >
字串長度	.size() or .length()
存取字元	s[i]
取子字串	.substr(start, len)
找字串	.find("sub")
清空字串	.clear()
加到末尾	.push_back('x') , .append(...)

字串特性補充

```
// Concatenation
s3= "Watermelon";
cout << "\nConcatenation" << "\n--<< endl;

s3=s5+ "and" + s2+ "juice";// Apple and Banana juice
cout << "s3 is now: " << s3 << endl; // Apple and Banana Juice

s3 = "nice" + "cold" + s5 + "juice";// Compiler error
//因為C String不能用"+"連接，要用strcat()
```