

Section 19 IO & Streams

Streams and I/O（資料流與輸入輸出）

C++ 中的 I/O 系統基於 **streams**（資料流）的概念來進行輸入與輸出操作。這提供了一個抽象的機制，使程式碼不需直接處理底層裝置細節即可與多種輸入輸出來源互動（例如鍵盤、檔案、螢幕等）。

◆ 1. Stream 是什麼？

- **Stream** 是一個位元序列（Sequence of bytes），在執行期間自動處理資料的流動方向。
- 以統一的方式處理 **文字（text）** 與 **二進位（binary）** 的資料。

◆ 2. 輸入與輸出資料流的方向

- **Input Stream（輸入流）**：提供資料給程式。
 - 例如 `std::cin` 從鍵盤讀取輸入。
- **Output Stream（輸出流）**：接收資料並輸出到裝置。
 - 例如 `std::cout` 輸出到螢幕。

◆ 3. 與裝置無關的設計（Device Independent）

- Streams 提供統一的介面，無論輸入來源是鍵盤、檔案或網路，程式介面不變。
- 這種抽象化使程式具備良好的可攜性與可維護性。

Stream Manipulators（資料流格式控制器）

在輸入輸出操作中，我們有時希望控制輸出的格式，例如設定欄寬、對齊方式、浮點數精度等。C++ 提供了兩種方式達成這些目的：

◆ 1. 成員函式（Member Functions）

- 使用 stream 物件的方法來改變狀態：

```
std::cout.width(10); // 將下一次輸出的欄寬設為 10
```

◆ 2. Manipulators（操作子/格式控制子）

- 使用 `<iomanip>` 標頭檔中的 **操控函式**（函式物件）來達到相同效果，更常見於輸出串接操作中：

```
#include <iomanip>
std::cout << std::setw(10) << 123; // 輸出欄寬為10的數字
```

◆ 3. 使用場景

- 可用於 **輸入與輸出流**。
- 有些 manipulator 僅在作用當下生效（如 `std::setw()`），有些會影響整個 stream 狀態（如 `std::fixed`，`std::setprecision()`）。


◆ 4. 常見 Manipulators（來自 `<iomanip>`）

Manipulator	說明
<code>std::setw(n)</code>	設定欄寬為 n（只作用於下一個輸出）
<code>std::setprecision(n)</code>	設定浮點數精度為 n 位數
<code>std::fixed</code>	使用固定小數點格式
<code>std::scientific</code>	使用科學記號格式
<code>std::left</code>	左對齊
<code>std::right</code>	右對齊
<code>std::showpoint</code>	顯示小數點與末尾的 0
<code>std::boolalpha</code>	將 bool 顯示為 true / false

◎ Stream Manipulators - Boolean

C++ 預設情況下，`bool` 型別的輸出會顯示為 `0` 或 `1`。但可以使用 manipulators 來改變這個格式：

Manipulator	說明
std::boolalpha	將 true / false 以文字形式顯示
std::noboolalpha	回復為數值形式 (1/0)


 範例：

```
#include <iostream>
int main() {
    std::cout << std::boolalpha << true << " " << false << "\n"; // true false
    std::cout << std::noboolalpha << true << " " << false << "\n"; // 1 0
}
```

Stream Manipulators - Integers

整數的輸出格式也可以透過 manipulators 控制，包括數字進位法與符號顯示：

Manipulator	說明
std::dec	十進位 (預設)
std::hex	十六進位
std::oct	八進位
std::showbase	顯示基底 (0x for hex , 0 for oct)
std::noshowbase	不顯示基底
std::showpos	顯示正號 (+)
std::noshowpos	不顯示正號 (預設)

 範例：

```
#include <iostream>
int main() {
    int value = 42;
    std::cout << std::hex << value << "\n"; // 2a
    std::cout << std::showbase << value << "\n"; // 0x2a
}
```

```
std::cout << std::dec << std::showpos << value << "\n"; // +42
}
```



Stream Manipulators - Floating Point

浮點數的格式可用下列 manipulators 控制其顯示方式與精度：

Manipulator	說明
<code>std::fixed</code>	使用固定小數點格式（例如 123.45）
<code>std::scientific</code>	使用科學記號格式（例如 1.2345e+02）
<code>std::setprecision(n)</code>	設定顯示的總精度或小數位數（依格式而定）
<code>std::showpoint</code>	即使小數部分為 0 也顯示小數點（例如 123.0）
<code>std::noshowpoint</code>	隱藏不必要的小數點（預設）

 範例：


```
#include <iostream>
#include <iomanip>
int main() {
    double pi = 3.14159;
    std::cout << std::fixed << std::setprecision(2) << pi << "\n"; // 3.14
    std::cout << std::scientific << std::setprecision(3) << pi << "\n"; // 3.142e+00
    std::cout << std::showpoint << 10.0 << "\n"; // 10.000
}
```



Stream Manipulators - Align and Fill（對齊與填充）

可以使用 manipulators 調整輸出的對齊方式與填充字元，常見於表格、報表格式：

Manipulator	說明
<code>std::setw(n)</code>	設定欄寬為 n
<code>std::left</code>	左對齊
<code>std::right</code>	右對齊（預設）
<code>std::internal</code>	符號靠左，數字靠右（常用於數字）
<code>std::setfill(c)</code>	將空白填滿為指定字元 c

 範例：

```
#include <iostream>
#include <iomanip>
int main() {
    std::cout << std::setw(10) << std::right << 42 << "\n";    // "    42"
    std::cout << std::setw(10) << std::left << 42 << "\n";      // "42    "
    std::cout << std::setw(10) << std::setfill('*') << 42 << "\n"; // "*****42"
}
```

manipulator function（操控器函式）的「生效範圍」：

一、短效 manipulators（只影響下一筆輸出）

這類 manipulator 的效果 只作用於下一次的輸出或輸入，例如：

Manipulator	生效範圍
<code>std::setw()</code>	只對下一次輸出生效
<code>std::setfill()</code>	持續生效，直到被改變為止
<code>std::setprecision()</code>	持續生效

 範例：

```
std::cout << std::setw(10) << 123 << 456;
// 結果： "    123456"
```

```
// `123` 受欄寬限制，但 `456` 不會
```

✅ 二、長效 manipulators（會改變 stream 狀態）

這些 manipulator 實際上會修改輸出串流內部的格式旗標（format flags），影響後續所有輸出，直到被其他 manipulator 或操作還原為止：

Manipulator	生效範圍
std::fixed	持續生效，直到設為 std::scientific 或其他
std::scientific	同上
std::showpos	持續生效
std::showpoint	持續生效
std::boolalpha	持續生效
std::hex, std::dec, std::oct	持續生效
std::left, std::right, std::internal	持續生效

📌 範例：

```
std::cout << std::fixed << std::setprecision(2);
std::cout << 3.14159 << "\n"; // 輸出：3.14
std::cout << 2.71828 << "\n"; // 輸出：2.72
// `fixed` 和 `setprecision` 持續有效
```

📁 Input Files – fstream and ifstream

在 C++ 中，處理檔案輸入的最常見方式就是使用 `<fstream>` 標頭檔裡的 `ifstream` 或 `fstream` 類別。這些類別繼承自 `istream`，可讓你像操作 `std::cin` 一樣，直接從檔案中讀取資料。

◆ 步驟 1：#include <fstream>

首先，要使用檔案輸入功能，你需要引入以下標頭檔：

```
#include <fstream>
```

◆ 步驟 2：宣告 ifstream 或 fstream 物件

```
std::ifstream inFile;    // 僅供輸入  
std::fstream fileStream; // 可用於輸入與輸出（需指定模式）
```

- ifstream 是專門用來讀取檔案的類別。
- fstream 則可同時用於輸入與輸出，但需搭配 mode 指定讀/寫權限。

◆ 步驟 3：連接檔案（開啟檔案）

你可以用以下方式開啟檔案：

```
inFile.open("data.txt"); // 用預設模式開啟以讀取
```

或直接在建構時給檔名：

```
std::ifstream inFile("data.txt");
```

✓ **小提醒：** 檔案開啟失敗時，inFile 會處於錯誤狀態，應該檢查：

```
if (!inFile) {  
    std::cerr << "檔案無法開啟！\n";  
}
```

◆ 步驟 4：透過 stream 讀取資料

檔案開啟後，可以用和 cin 一樣的方式讀取資料：

```
int num;
std::string word;

inFile >> num >> word;
```

📌 或使用 `getline` 逐行讀取：

```
std::string line;
while (std::getline(inFile, line)) {
    std::cout << line << "\n";
}
```

◆ 步驟 5：關閉檔案（Good Practice）

完成讀取後，記得使用 `.close()` 關閉檔案：

```
inFile.close();
```

雖然離開區塊時 destructor 會自動關閉檔案，但明確關閉是一種良好習慣，尤其是在開啟多個檔案或需寫入的情況下。

📌 完整範例

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

int main() {
    std::ifstream inFile("example.txt");

    if (!inFile) {
        std::cerr << "檔案無法開啟！\n";
        return 1;
    }
}
```



```

std::string line;
while (std::getline(inFile, line)) {
    std::cout << line << "\n";
}

inFile.close();
return 0;
}

```

Output Files – `fstream` and `ofstream`

在 C++ 中，輸出資料到檔案常使用 `<fstream>` 標頭中的 `ofstream` 或 `fstream` 類別。這些類別繼承自 `ostream`，可讓你像使用 `std::cout` 一樣，將資料寫入檔案。

◆ 步驟 1：`#include <fstream>`

使用檔案輸出功能前，必須引入標頭檔：

```
#include <fstream>
```

◆ 步驟 2：宣告 `ofstream` 或 `fstream` 物件

```

std::ofstream outFile; // 僅供輸出
std::fstream fileStream; // 可用於輸入與輸出（需指定模式）

```

- `ofstream`：專門用於輸出資料到檔案。
- `fstream`：可同時用於讀寫，但必須指定模式（如 `std::ios::out`）。

◆ 步驟 3：連接檔案（開啟檔案寫入）

可以用 `.open()` 函式，也可在建構時指定檔名：

```
outFile.open("output.txt");    // 明確開啟
std::ofstream outFile("output.txt"); // 建構時開啟
```

💡 注意：如果檔案存在，預設會被清空（truncated）。

✅ 若要避免覆蓋，可加 `std::ios::app` 模式：

```
std::ofstream outFile("output.txt", std::ios::app); // 追加模式
```

◆ 步驟 4：透過 stream 寫入資料

與 `std::cout` 相同，使用 `<<` 將資料寫入檔案：

```
outFile << "Hello, file!" << "\n";
int x = 42;
outFile << "x = " << x << std::endl;
```

也可用 `put()` 寫入單一字元：

```
outFile.put('A');
```

◆ 步驟 5：關閉檔案（Good Practice）

寫入完成後，請使用 `.close()` 關閉檔案：

```
outFile.close();
```

雖然檔案物件在離開作用範圍後會自動關閉，但明確呼叫 `.close()` 能让你更早釋放資源，特別是在多檔案操作時很重要。

完整範例

```

#include <iostream>
#include <fstream>

int main() {
    std::ofstream outFile("output.txt");

    if (!outFile) {
        std::cerr << "檔案無法開啟！\n";
        return 1;
    }

    outFile << "這是寫入檔案的文字。" << std::endl;
    outFile << "數值： " << 123 << "\n";

    outFile.close();
    return 0;
}

```

常見的檔案串流方法（適用於 ifstream / ofstream）

以下這些成員函式可用於檢查檔案狀態、控制讀寫位置、或處理錯誤：

狀態檢查與控制

方法 / 屬性	說明
<code>.is_open()</code>	檢查檔案是否成功開啟。回傳 <code>true</code> 或 <code>false</code> 。
<code>.good()</code>	檢查是否處於「好狀態」（沒有錯誤）。
<code>.eof()</code>	是否到達檔案結尾（End Of File）。
<code>.fail()</code>	是否發生失敗（通常是格式錯誤或開啟失敗）。
<code>.bad()</code>	是否發生嚴重錯誤（例如硬碟損壞、系統錯誤）。
<code>.clear()</code>	重設錯誤狀態（清除 <code>.fail()</code> 或 <code>.bad()</code> 狀態）。

📌 範例：

```
std::ifstream inFile("data.txt");
if (inFile.is_open()) {
    if (inFile.fail()) {
        std::cerr << "讀取失敗！\n";
        inFile.clear(); // 重設錯誤狀態
    }
}
```

🕒 位置控制（適用於 ifstream / ofstream / fstream）

用來查詢或改變檔案的讀寫位置（適用於隨機存取）：

方法	說明
.tellg()	傳回目前「讀取位置」的位元組索引（ifstream）
.tellp()	傳回目前「寫入位置」的位元組索引（ofstream）
.seekg(pos)	將讀取位置移動到 pos
.seekp(pos)	將寫入位置移動到 pos
.seekg(offset, dir)	相對於某個方向移動讀取位置（例如 std::ios::beg）
.seekp(offset, dir)	相對於某個方向移動寫入位置

📌 範例（移動到檔案開頭）：

```
inFile.seekg(0, std::ios::beg); // 從檔案開頭重新讀取
```

📁 開啟與關閉檔案


方法	說明
.open(filename)	開啟檔案（建構之後也能開啟）

方法	說明
<code>.close()</code>	關閉檔案

檔案開啟模式（`open()` 的第二參數）

當你使用 `.open()` 時可以指定開啟模式，這些常數定義於 `std::ios` 中：

模式	說明
<code>std::ios::in</code>	讀取模式（預設 <code>ifstream</code> ）
<code>std::ios::out</code>	寫入模式（預設 <code>ofstream</code> ）
<code>std::ios::app</code>	附加模式（append）
<code>std::ios::ate</code>	移動到檔案尾後再操作
<code>std::ios::trunc</code>	開啟時清空原始檔案
<code>std::ios::binary</code>	以二進位格式開啟

 範例（以讀寫模式開啟）：

```
std::fstream file("data.txt", std::ios::in | std::ios::out);
```

小提醒：`.getline()` vs `std::getline()`

- `.getline(char* buffer, size_t)` 是 `istream` 的成員函式，只能讀 C-string
- `std::getline(istream, std::string)` 是標準函式，可直接用在 `std::string` 上

總結圖表

功能	方法 / 屬性
狀態檢查	<code>is_open()</code> , <code>eof()</code> , <code>fail()</code>
錯誤清除	<code>clear()</code>

功能	方法 / 屬性
位置控制	seekg() , seekp() , tellg() , tellp()
開關檔案	open() , close()

Using String Streams – stringstream , istream , ostream

C++ 提供了字串串流類別，可以將 `std::string` 當作輸入或輸出的來源/目標，就像操作檔案或 `cin / cout` 一樣，這對於字串的格式化處理特別有用。

這些類別都來自 `<sstream>` 標頭：

類型	說明
<code>std::istream</code>	用來「從 string 讀取資料」（像 <code>cin</code> ）
<code>std::ostream</code>	用來「寫資料進 string」（像 <code>cout</code> ）
<code>std::stringstream</code>	同時可讀可寫

◆ 步驟 1：#include <sstream>

```
#include <sstream>
```

◆ 步驟 2：宣告 stringstream , istream , ostream 物件

```
std::istream inputStream;
std::ostream outputStream;
std::stringstream ioStream;
```

◆ 步驟 3：連接 `std::string`（初始化或取得內容）

- 你可以在建構時或 `.str()` 方法中設定或取得底層字串：

```
std::string data = "123 456";
std::istringstream iss(data);    // 讀取模式
std::ostringstream oss;         // 寫入模式
oss << "Hello, " << 2025;       // 將資料寫入 stringstream
std::string result = oss.str();  // 取得寫入後的字串
```

◆ 步驟 4：使用格式化 I/O 操作字串流

✓ `istringstream`（讀取字串）

```
std::string line = "42 3.14 hello";
std::istringstream iss(line);
int x;
double pi;
std::string word;
iss >> x >> pi >> word; // 就像從 cin 讀入一樣
```

✓ `ostringstream`（寫入字串）

```
std::ostringstream oss;
oss << "Result: " << 99 << ", OK\n";
std::string output = oss.str(); // 轉換成 std::string 使用
```

✓ `stringstream`（可讀可寫）

```
std::stringstream ss;
ss << "C++ 2025";
std::string lang;
int year;
ss >> lang >> year; // 可以讀回剛剛寫進去的資料
```

stringstream 的應用場景

應用	說明
字串分割	用空格自動拆解字串成數值或詞彙
字串轉數字	比 <code>std::stoi()</code> 更穩定和通用
格式化輸出成字串	可控制精度、欄寬後儲存為 <code>std::string</code>
單行輸入解析	用在 <code>std::getline()</code> 後處理內容

範例：拆解使用者輸入的數字字串

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

int main() {
    std::string input = "100 3.14 OpenAI";
    std::istringstream iss(input);

    int i;
    double d;
    std::string word;
    iss >> i >> d >> word;

    std::cout << "Int: " << i << "\n";
    std::cout << "Double: " << d << "\n";
    std::cout << "Word: " << word << "\n";

    return 0;
}
```