IO

王淇 (LittleCube)

台南一中資訊社 TFCIS

2021.9.22

Outline

運作原理

- Input/Output

運作原理

- Input/Output
- Standard streams

運作原理

- Input/Output
- Standard streams
- Redirection

運作原理

- 字元 $\leftrightarrow$ 程式

運作原理

- 字元 $\leftrightarrow$ 程式
- 人　 $\leftrightarrow$ 車站

運作原理

- 字元 $\leftrightarrow$ 程式
- 人　　$\leftrightarrow$ 車站

- 太慢了！

運作原理

- 字元 $\leftrightarrow$ 緩衝區 $\leftrightarrow$ 程式

運作原理

- 字元 $\leftrightarrow$ 緩衝區 $\leftrightarrow$ 程式
- 人 $\leftrightarrow$ 接駁車 $\leftrightarrow$ 車站

運作原理

- 字元 $\leftrightarrow$ 緩衝區 $\leftrightarrow$ 程式
- 人 $\leftrightarrow$ 接駁車 $\leftrightarrow$ 車站

- 緩衝區 (Buffer) 可以加速輸入輸出

運作原理

- 字元 $\leftrightarrow$ 緩衝區 $\leftrightarrow$ 程式
- 人 $\leftrightarrow$ 接駁車 $\leftrightarrow$ 車站

- 緩衝區 (Buffer) 可以加速輸入輸出
- 慢是慢在存取 IO 的地方，不是 IO 本身
- 缺點是顯示未必與實際運算同步

運作原理

- **stdin** (**St**an**d**ard **In**put)

運作原理

- **stdin** (**St**and**d**ard **In**put)
- **stdout** (**St**and**d**ard **Out**put)

運作原理

- **stdin** (**St**and**d**ard **In**put)
- **stdout** (**St**and**d**ard **Out**put)
- **stderr** (**St**and**d**ard **Err**or)

運作原理

- **stdin** (**St**and**d**ard **In**put)
- **stdout** (**St**and**d**ard **Out**put)
- **stderr** (**St**and**d**ard **Err**or)

- **stdout** 跟 **stderr** 通常會一起顯示，但意義不同

IO

- 接下來介紹的都在 `std` 底下，請加上 `using namespace std;`
- C style IO 在 `<cstdio>` 裡
- C++ style IO 在 `<iostream>` 裡

C style IO

```
1 int scanf( const char* format, ... );
```

C style IO

```
1 int scanf( const char* format, ... );
```

format 的規則：

C style IO

```
1 int scanf( const char* format, ... );
```

format 的規則：

- 非空白字元 $\Rightarrow$ 匹配一樣的字元，失敗就不會讀

C style IO

```
1 int scanf( const char* format, ... );
```

format 的規則：

- 非空白字元 $\Rightarrow$ 匹配一樣的字元，失敗就不會讀
- 空白字元 $\Rightarrow$ 匹配空白、換行

C style IO

```
1 int scanf( const char* format, ... );
```

format 的規則：

- 非空白字元 ⇒ 匹配一樣的字元，失敗就不會讀
- 空白字元 ⇒ 匹配空白、換行
- % ⇒ 標示讀入變數的格式

C style IO

```
1 int scanf( const char* format, ... );
```

format 的規則：

- 非空白字元 ⇒ 匹配一樣的字元，失敗就不會讀
- 空白字元 ⇒ 匹配空白、換行
- % ⇒ 標示讀入變數的格式
- https://en.cppreference.com/w/cpp/io/c/fscanf

C style IO

```
1  scanf("%d", &i);     // int
2  scanf("%lld", &i);   // long long int
3  scanf("%f", &i);     // float
4  scanf("%lf", &i);    // double
5  scanf("%c", &i);     // char
6  scanf("%s", &i);     // string (char[])
```

C style IO

```
1  scanf("%d", &i);     // int
2  scanf("%lld", &i);   // long long int
3  scanf("%f", &i);     // float
4  scanf("%lf", &i);    // double
5  scanf("%c", &i);     // char
6  scanf("%s", &i);     // string (char[])
```

- 變數記得加 &

C style IO

```
1  scanf("%d", &i);     // int
2  scanf("%lld", &i);   // long long int
3  scanf("%f", &i);     // float
4  scanf("%lf", &i);    // double
5  scanf("%c", &i);     // char
6  scanf("%s", &i);     // string (char[])
```

- 變數記得加 &
- 讀入的切割是用空白字元分開的

C style IO

```
1 scanf("%d", &i);     // int
2 scanf("%lld", &i);   // long long int
3 scanf("%f", &i);     // float
4 scanf("%lf", &i);    // double
5 scanf("%c", &i);     // char
6 scanf("%s", &i);     // string (char[])
```

- 變數記得加 &
- 讀入的切割是用空白字元分開的
- 回傳成功讀到多少變數

C style IO

- 想讀整行？

C style IO

- 想讀整行？

```
1 scanf("%[^\n]s",name);
```

C style IO

```
1 int printf( const char* format, ... );
```

C style IO

```
1 int printf( const char* format, ... );
```

format 的規則：

C style IO

```
1 int printf( const char* format, ... );
```

format 的規則：

- % ⇒ 標示輸出變數的格式

C style IO

```
1 int printf( const char* format, ... );
```

format 的規則：

- % ⇒ 標示輸出變數的格式
- https://en.cppreference.com/w/cpp/io/c/fprintf

C style IO

```
1 printf("%d", 1);          // 1
2 printf("%lld", 1);
3 printf("%.3f", 1);        // 1.000
4 printf("%.4f", 1.23456);// 1.2346
5 printf("%c", 65);          // A
6 printf("Hello World");   // Hello World
```

C style IO

```
1 printf("%d", 1);          // 1
2 printf("%lld", 1);
3 printf("%.3f", 1);        // 1.000
4 printf("%.4f", 1.23456);// 1.2346
5 printf("%c", 65);          // A
6 printf("Hello World");   // Hello World
```

- 回傳成功輸出多少字元

C style IO

- 讀檔

```
1 freopen("input.txt", "r", stdin)
2 freopen("output.txt", "w", stdout)
```

C++ style IO

- 與 C 不同的是，C++ 的 IO Stream 是物件，採用運算元處理

C++ style IO

- 與 C 不同的是，C++ 的 IO Stream 是物件，採用運算元處理

```
1  int a, b;
2  char c;
3  double d;
4
5  cin >> a >> b;
6  cin >> c;
7  cin >> d;
```

C++ style IO

- 與 C 不同的是，C++ 的 IO Stream 是物件，採用運算元處理

```
1 int a, b;
2 char c;
3 double d;
4
5 cin >> a >> b;
6 cin >> c;
7 cin >> d;
```

- 會按照順序

C++ style IO

- 與 C 不同的是，C++ 的 IO Stream 是物件，採用運算元處理

```
1  int a, b;
2  char c;
3  double d;
4
5  cin >> a >> b;
6  cin >> c;
7  cin >> d;
```

- 會按照順序
- 以空白分開

C++ style IO

- 與 C 不同的是，C++ 的 IO Stream 是物件，採用運算元處理

```
1 int a, b;
2 char c;
3 double d;
4
5 cin >> a >> b;
6 cin >> c;
7 cin >> d;
```

- 會按照順序
- 以空白分開
- 讀取成功或失敗可以用 `if` 判斷

C++ style IO

- 與 C 不同的是，C++ 的 IO Stream 是物件，採用運算元處理

```
1 int a, b;
2 char c;
3 double d;
4
5 cin >> a >> b;
6 cin >> c;
7 cin >> d;
```

- 會按照順序
- 以空白分開
- 讀取成功或失敗可以用 `if` 判斷
- 可以運算元重載

C++ style IO

- 想讀整行？

C++ style IO

- 想讀整行？

```
1 string s;
2 getline(cin, s);
```

C++ style IO

- 想讀整行？

```
1 string s;
2 getline(cin, s);
```

- 注意 `cin` 完會放一個換行在尾端，切換時要讀兩次才會對

C++ style IO

- 想讀整行？

```
1 string s;
2 getline(cin, s);
```

- 注意 `cin` 完會放一個換行在尾端，切換時要讀兩次才會對
- 另外有 `cin.getline()`，但使用較為麻煩

C++ style IO

- 在 `namespace std` 底下

C++ style IO

- 在 namespace std 底下

```
1 cout << 1 << 2;           // 12
2 cout << 1 << ' ' << 2;    // 1 2
3 cout << -1 << -2;         // -1-2
4 cout << "He" << "llo";    // Hello
```

C++ style IO

- 在 namespace std 底下

```
1 cout << 1 << 2;           // 12
2 cout << 1 << ' ' << 2;    // 1 2
3 cout << -1 << -2;         // -1-2
4 cout << "He" << "llo";    // Hello
```

- 會按照順序

C++ style IO

- 在 namespace std 底下

```
1 cout << 1 << 2;           // 12
2 cout << 1 << ' ' << 2;    // 1 2
3 cout << -1 << -2;         // -1-2
4 cout << "He" << "llo";    // Hello
```

- 會按照順序
- 可以運算元重載

C++ style IO

- 在 namespace std 底下

```
1 cout << 1 << 2;           // 12
2 cout << 1 << ' ' << 2;    // 1 2
3 cout << -1 << -2;         // -1-2
4 cout << "He" << "llo";    // Hello
```

- 會按照順序
- 可以運算元重載
- cerr 也是一樣的用法

C++ style IO

小數點要怎麼調控？

C++ style IO

小數點要怎麼調控？

IO Manipulator
在 `<iomanip>` 裡

C++ style IO

小數點要怎麼調控？

IO Manipulator
在 `<iomanip>` 裡

- `fixed` 印出固定位數的小數點

C++ style IO

小數點要怎麼調控？

IO Manipulator
在 `<iomanip>` 裡

- `fixed` 印出固定位數的小數點
- `setprecision(int n)` 設定印出 n 位數的小數點

C++ style IO

小數點要怎麼調控？

IO Manipulator
在 `<iomanip>` 裡

- `fixed` 印出固定位數的小數點
- `setprecision(int n)` 設定印出 n 位數的小數點
- `endl` 換行並清空 Buffer

C++ style IO

小數點要怎麼調控？

IO Manipulator
在 `<iomanip>` 裡

- `fixed` 印出固定位數的小數點
- `setprecision(int n)` 設定印出 n 位數的小數點
- `endl` 換行並清空 Buffer
- `flush` 清空 Buffer

C++ style IO

小數點要怎麼調控？

IO Manipulator
在 `<iomanip>` 裡

- `fixed` 印出固定位數的小數點
- `setprecision(int n)` 設定印出 n 位數的小數點
- `endl` 換行並清空 Buffer
- `flush` 清空 Buffer

```
1 cout << fixed << setprecision(6);    // (nothing)
2 cout << 1.2345678 << '\n';           // 1.234568
3 cout << "1.2345678" << '\n';         // 1.2345678
4 cout << 1.0 << '\n';                 // 1.0000000
```

C++ style IO

- 讀檔

```
1  ifstream fin;
2  fin.open("input.txt");
3  // ifstream fin("input.txt");
4  fin.close();
5
6  ofstream fout;
7  fout.open("output.txt");
8  //ofstream fout("output.txt");
9  fout.close();
```

比較

- 直覺性

比較

- 直覺性
  C++ style (`cin, cout`)

比較

- 直覺性
  C++ style (`cin, cout`)
- 簡潔

比較

- 直覺性

  C++ style (`cin, cout`)
- 簡潔

  C style (`scanf, printf`)

```
1 printf("%d %d %d %d %d\n", a, b, c, d, e);
2 cout << a << ' ' << b << ' ' << c << ' ' << d << ' ' <<
      e << '\n';
```

比較

- 直覺性

  C++ style (`cin, cout`)
- 簡潔

  C style (`scanf, printf`)

```
1 printf("%d %d %d %d %d\n", a, b, c, d, e);
2 cout << a << ' ' << b << ' ' << c << ' ' << d << ' ' <<
      e << '\n';
```

- 速度

比較

- 直覺性

  C++ style (`cin, cout`)

- 簡潔

  C style (`scanf, printf`)

```
1 printf("%d %d %d %d %d\n", a, b, c, d, e);
2 cout << a << ' ' << b << ' ' << c << ' ' << d << ' ' <<
      e << '\n';
```

- 速度

  C style (`scanf, printf`) 比較快，
  但為什麼？

比較

C++ style (`cin`, `cout`) 同步了 **C, C++ 兩種 IO** 跟**輸入輸出**

比較

C++ style (`cin`, `cout`) 同步了 **C, C++ 兩種 IO 跟輸入輸出**

- `ios::sync_with_stdio(0)` 解除 **C, C++ 兩種 IO** 的同步

比較

C++ style (`cin`, `cout`) 同步了 **C, C++ 兩種 IO 跟輸入輸出**

- `ios::sync_with_stdio(0)` 解除 **C, C++ 兩種 IO** 的同步
  加了這行就不能混用兩種輸出

比較

C++ style (`cin, cout`) 同步了 **C, C++ 兩種 IO** 跟**輸入輸出**

- `ios::sync_with_stdio(0)` 解除 **C, C++ 兩種 IO** 的同步
  加了這行就不能混用兩種輸出
- `cin.tie(0)` 解除**輸入輸出**的同步

比較

C++ style (`cin, cout`) 同步了 **C, C++ 兩種 IO** 跟**輸入輸出**

- `ios::sync_with_stdio(0)` 解除 **C, C++ 兩種 IO** 的同步
  加了這行就不能混用兩種輸出
- `cin.tie(0)` 解除**輸入輸出**的同步
  加了這行，輸入輸出並不再代表實際運作順序

比較

C++ style (`cin, cout`) 同步了 **C, C++ 兩種 IO** 跟**輸入輸出**

- `ios::sync_with_stdio(0)` 解除 **C, C++ 兩種 IO** 的同步
  加了這行就不能混用兩種輸出
- `cin.tie(0)` 解除**輸入輸出**的同步
  加了這行，輸入輸出並不再代表實際運作順序

這樣一來，兩個就會差不多快