Class

※ C++ 的 struct 也能像這，自行取修飾詞

```
class Myclass{
private:
    int data;
    int *a;
public:
    int num;
    explicit Myclass(int x, int*y, int z): data(x), a(y), num(z){}
or  explicit Myclass(int x, int*y, int z){
        data = x; a = y; num = z;
    }
    void show() const{
        std::cout<<data;
    }
protected:
    int secret;
};
```

① private 下的變數只能提供給 Myclass 的 member functions 使用

正確初始化方法

Myclass class1(a, &b, 3);

② (1) 函數名稱後加 const 表該 function 不能修改成員變數 (namely "data")

此類 function 只能被 const 物件呼叫

③ private 下的此式可以被專案中的其他 functions 使用 (ex: "main()")

private 只能在當前類別中使用

④ explicit 可有可無，作用是可防止隱式型別轉換

Myclass class1 = 6;

→ 6 將會被轉換成 class 型別，但因為有 explicit 所以會報錯

⑤ protected：子類別可以用，但外面不能用

static 宣告在 public 中的 static 變數的值，被屬於此 class 的所有物件所共用

Myclass::count = 3; ∴可以用 "::" 修改「這個 class」的值（∵共用）

succeed ※A：子類；B：父類

class A: public B 從外面可存取 B 的 public

class A: protected B 子類別可存取父類的 public，protected，但外面不行

class A: private B 子類別可存取父類的 public，protected，但外面不行，且不知道你繼承

other

class A: public B　class B: public C

此時 A 等同於繼承 C，A可以直接使用C的public、protected

class A: public B, public C　class B: public D　class C: public D

此時若從A呼叫D的函式，系統會不知道要用B繼承來的D還是C的D，會產生問題

String

Input

```
getline(cin, s1, delimeter);
```

Alan Chen ⇒ Alan C

delimeter is 'h'

↳ delimeter is optional

Claim

```
string text {“Hello”};
text = “Flirt”;
string text1 (8, ‘a’);
```

① string是一種資料型態，可以直接賦值

② two arguments的宣告方法是錯用()

↳只能是字元

Function

※ start 是 index

```
str.size();
str.length();
```

① 測量string的長度

② 尾字元沒有'\0'

```
str.append(5, ‘!’);
str.append(“abcde”, 3);
```

⇒ append "!!!!!"　① 有支援 C-style string

⇒ append "abc"　② 回傳 string&

```
str.assign(source, start, nums);
```

字串複製，且可以從指定位置開始及指定長度

```
string s (“Hello “);
string s1 (s + “Yik” + “安安”);
```

① string 的+運算

② ∵"Hello"是指向'h'的 char 指標 ∴ "Hello"+"Yik"行

```
for (char c: str) /*code*/;
```

遍歷字串

```
str1 > str2
str1.compare(start1, nums1, str2, start2, nums2);
```

① 要麻只省略start2、nums，或 start1、nums1、start2、nums，不然就都到省略

② nums為比較的字元數

str.at(2) = 'w';

① 若超出索引值，會拋出 std::out_of_range

② 功能與 str[2] 一樣

str.**substr**(start, nums); 返回從 start 開始指定字元數的 string

str1.**swap**(str2); 交換兩 string 的值

str.**find**(string) str.**rfind**(string) 從 index 0 / index n 找，回傳 string 第一次的位置

str.**find_first_of**(string) string 中任意字元在 str 中第一次出現的位置

str.**find_last_of**(string) string 中任意字元在 str 中最後一次出現的位置

str.**find_first_not_of**(string) 從 index 0 找，不出現在 string 中的第一個字元的位置

str.**erase**(index) Remove 從指定 index 開始的所有字元

str.**replace**(index, n, string) 從指定 index 開始，替代 n 個字元為 string

str.**replace**(index, n, string, start, end) 是替換替代的 string 有範圍 → 含 end

str.**insert**(index, string, start, end) 將指定範圍的字串，插入指定 index

converted to pointer-based char* string

char *p{new char[len+1]};
str.**copy**(p, len, pos); p[len] = '\0';

① 記得 char* based 的 string 有 '\0'

② copy() 用來將 string 內字元複製至 char 陣列

str.c_str() 直接回傳 char* based 的 string

Pair

※ include <utility>

basic (p.first , p.second) 比較大小時先比 first

making an object make_pair(false, “Alan”)

pair<bool, string>(false, “Alan”)

Map

※ include <map> ※ insert/search/delete ⇒ O(logn)

basic ① 一個 key ⟷ 一個 value ② 預設用 < 來排序 key ③ 元素為 pair

map[key] = value value = map[key] 若無該 key，則 value = 0

map.at(key) map.count(key) 回傳是否有該 key，值為 0 or 1

map.find(key) 回傳 iterator，找不到回傳 end()

map.erase(key) map.clear() 清空 map map.size()

map.insert({key, score}) ① 若該 key 存在，則不會覆蓋值 ② 若該 key 不存在，則新增 pair

不管怎樣都會回傳該 pair 的 iterator

customized sort ※ 要用特殊方法才能 sort value

map<int, string, **greater<int>**> m; key 由大到小

map<string, bool, [](const string& a, const string& b){return a.length() < b.length();}> m; 使用 lambda 自訂義排序 key

vector<pair<string, int>> sorted_M(M.begin(), M.end());

sort(sorted_M.begin(), sorted_M.end(), [](auto& a, auto& b) {return a.second > b.second;});

① 先將 map 複製進 vector，再排序 value ② 非即時排序

vector

initialize

```
vector<vector<int>> m1;
vector<vector<int>> m2(3, vector<int>(4, 0));
```

```
vector<int> v1(3);
vector<int> v2(3, -1);
vector<int> v3{1,2,3,4,5};
vector<int> v4(v3.begin()+1, v3.end()-1);
vector<int> v5(v4);
```

也要用.begin(), .end()
改的是參考
arr也能拿來複製

⇒ {0,0,0}
⇒ {-1,-1,-1}
⇒ {1,2,3,4,5}
⇒ {2,3,4}
⇒ {2,3,4}

C style 用 begin(arr), end(arr)

assign

```
v1 = v2;
```

compare

```
v1 == v2
v1 != v2
```

basic

v.size()	回傳目前元素數量	v.begin()	傳回指向第一個元素的迭代器
v.empty()	回傳是否為空 vector (true/false)	v.end()	傳回指向「尾端 +1」的迭代器
v.capacity()	回傳目前分配的儲存空間 (容量)	v.rbegin()	反向迭代器：從尾到頭
v.max_size()	回傳 vector 最多可以存多少元素	v.rend()	反向結束 (在頭的前一個位置)
v.resize(n)	重新調整大小，可能補 0 或截斷		

resize(n, fill_value)
① vector 變長可直接填上 fill_value ② vector 變短會銷毀在 n 以外的 value

insert & delete

v.push_back(val)	尾端加入元素
v.pop_back()	移除最後一個元素
v.insert(it, val)	在指定位置插入元素
v.erase(it)	移除指定位置的元素
v.clear()	清空 vector 內容

access

v[i]	用索引存取元素 (不檢查範圍)
v.at(i)	存取元素，會檢查範圍 (會 throw)
v.front()	取得第一個元素
v.back()	取得最後一個元素
v.data()	回傳原始資料的指標 (int*)

※it 是 iterator 之型別與 pointer 不同

assign

v.assign(n, val)	將 vector 設為 n 個 val
v = other_vector	直接指派
v.swap(v2)	交換兩個 vector 的內容

others

v.shrink_to_fit()	釋放未使用的記憶體
v.reserve(n)	預留空間，避免頻繁重配記憶體
v.shrink_to_fit()	把 capacity 壓縮成剛好 size

Function fitting

Function overloading

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

當此式名稱相同，但參數不同，compiler 會使用 input parameter 的 number, types, and order 去選擇要執行的 function

Function template

```
template <typename T>
T max( T value 1, T value2, T value3 ){...}
```

存在 max.h 檔案裡，需 include，可避免自己不用的 type

Others

※ 60000000 equals to 60'000'000

setw(n); 輸出空白字元 ※ '1' ⇒ 49 , '7' ⇒ 55

safer random number generator # include <random> , <ctime>

```
default_random_engine engine{static_cast<unsigned int>(time(0))};
uniform_int_distribution<unsigned int> randomInt{1,6};
```

以上述方式設置後，以 randomInt(engine) 生成

```
int main(){
    int x;
    {
        int y;
    }
}
```

y是 main() 的 local variable

inline int foo();

將呼叫函式的地方直接替換為函式的實際程式碼
① 降低編譯效率
② 提升執行效能
③ 適合用於小型函式

```
int a = 3;
void foo{ int a = 2; cout << ::a << endl; }
```

當 local variable 和 global variable 有相同名稱，可用 '::' access

auto ① 一種 variable 修飾詞 ② 自動尋找適配的型別 ③ 省下打字的時間

making an object

```
return vector {1,2,3};
```

```
vector<pair<string, int>> V(map.begin(), map.end())
```

將 map 複製進 vector

duplicate # include <algorithm>

```
copy(start, end, new_start);
copy_n(start, n, new_start);
```

start , end , new_start 放 iterator or pointer 皆可

Others

new

```
int* p1 = new int;
char* p2 = new char[]{1,2,3};
vector<int>* v = new vector<int>{1,2,3};
```

delete

```
delete p1;
delete[] p2;
delete v;
```

structure binding

```
auto [it, success] = m.insert({'a', 90});
```

it 和 success 會直接對到回傳的 pair 的結構

```
for (auto& [name, score]: scores){...}
```

scores 是 map；name 和 score 直接對應 pair 的結構

sort()

```
sort(start, end, comp);
```

start 和 end 一定要用 nums.begin() 和 nums.end() 的運算式表示

```
bool comp(int i, int j){ return i > j; }
```

comp 省略，預設為升冪，也可自訂 ↳ 降冪

binary_search()

```
binary_search(start, end, comp);
```

如果是自訂義型別，則要在 comp() 動手腳

File

```
std::ofstream file(path, ios::app);
file << ...
```

↑此為 append 模式，預設為覆寫

```
std::ifstream file(path);
std::getline(std::file, line);
file >> line;
```

讀寫同時

```
std::fstream(path, ios::in | ios::out | ios::trunc);
```

↑沒有的話不清空

會清空檔案

Cast

static_cast<type>( )

static 用在已確保安全的轉型狀況

① 從子類別 → 基類別 ② void* → 具体指標型別 ③ enum → int

④ 希望避免隱式轉換可能的問題，and more readable

dynamic

Enum

```
enum class Color{ Red, Green, Blue };
```

① Red 預設為 0，其餘為依序遞增 ② 需轉型成 int 才能以整數顯示

```
enum class Color : short { Red, Green, Blue };
```

可指定型別的配合值的範圍

Array

#include <array>

不用每次都 initialize，可提升效能

initialize `static array<int, 5> arr{1, 2};` 也還是能用 C 的宣告方法

size_t ① it is unsigned ② 常用在 size 相關 `arr.size()` 會回傳 arr 的大小

range-based ※ 避免 segmentation fault ※ vector 也能這樣

`for (int item: items){}` `for (int& item: items){}` 右邊才能修改 items[] 的值

multi-dimensional array initialize `array<array<int, 4>, 3> arr{{{1,2,3}, {4,5,6}}};`

初始化二維以上要多一組{}

File

此為 append 模式，預設為覆寫

```
std::ofstream file(path, ios::app);
file << ...
```

```
std::ifstream file(path);
std::getline(std::file, line);
file >> line;
```

讀寫同時

沒有的話不清空

```
std::fstream(path, ios::in | ios::out | ios::trunc);
```

會清空檔案

Exception Handling

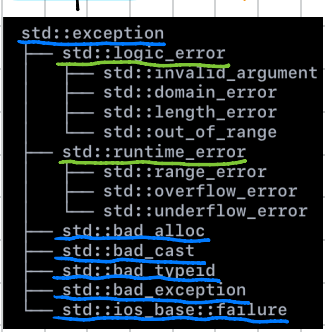
```
try{
    ...
}
catch(type variable){
    ...
}
```

① 一定要經由 throw，catch 才能接收到

② 可以 throw 任何類型的值或物件，以及標準例外類型

Exception

※ e.what() 可用來取得錯誤訊息字串




```
catch(out_of_range& e){
    ...
}
catch(...){
    ...
}
```

① exception 底下包含各種例外類型，而 exception 也是一種，∴它可以 catch 各種錯誤

② catch(...) 可捕捉沒列出的例外類型

```
# include <exception>
# include <stdexcept>
```

sort()

```
sort(start, end, comp);
```

start 和 end 一定要用 nums.begin() 和 nums.end() 的運算式表示

```
bool comp(int i, int j){ return i > j; }
```

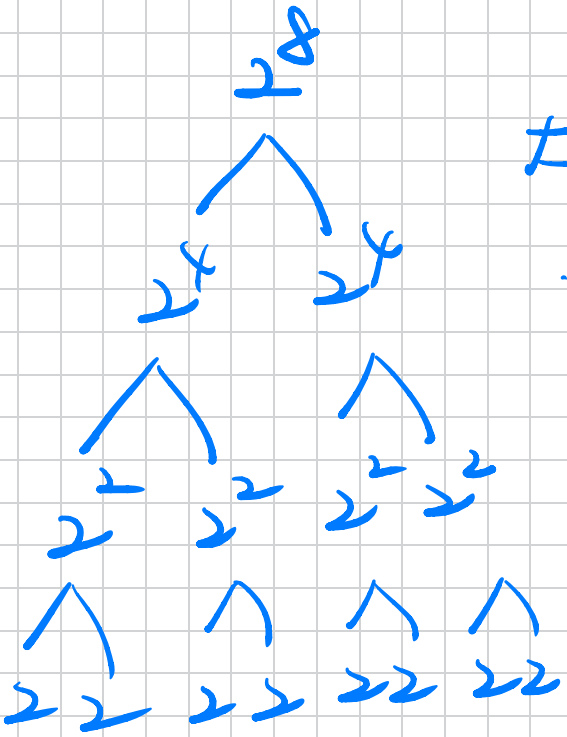
comp 省略，預設為升冪，也可自訂

↳ 降冪

binary_search()

```
binary_search(start, end, comp);
```

如果是自訂義型別，則要在comp() 動手腳即



大多数情况

iteration is more efficient

$2^8 = 2 \cdot 2^7$ 约呼叫8次

Standard Library header	Explanation
`<iostream>`	Contains function prototypes for the C++ standard input and output functions, introduced in Chapter 2, and is covered in more detail in Chapter 13, Stream Input/Output: A Deeper Look.
`<iomanip>`	Contains function prototypes for stream manipulators that format streams of data. This header is first used in Section 4.10 and is discussed in more detail in Chapter 13, Stream Input/Output: A Deeper Look.
`<cmath>`	Contains function prototypes for math library functions (Section 6.3).
`<cstdlib>`	Contains function prototypes for conversions of numbers to text, text to numbers, memory allocation, random numbers and various other utility functions. Portions of the header are covered in Section 6.7; Chapter 11, Operator Overloading; Class `string`; Chapter 17, Exception Handling: A Deeper Look; Chapter 22, Bits, Characters, C Strings and `structs`; and Appendix F, C Legacy Code Topics.
`<ctime>`	Contains function prototypes and types for manipulating the time and date. This header is used in Section 6.7.

Fig. 6.5 | C++ Standard Library headers. (Part 1 of 4.)

Standard Library header	Explanation
`<array>`, `<vector>`, `<list>`, `<forward_list>`, `<deque>`, `<queue>`, `<stack>`, `<map>`, `<unordered_map>`, `<unordered_set>`, `<set>`, `<bitset>`	These headers contain classes that implement the C++ Standard Library containers. Containers store data during a program's execution. The `<vector>` header is first introduced in Chapter 7, Class Templates `array` and `vector`; Catching Exceptions. We discuss all these headers in Chapter 15, Standard Library Containers and Iterators. `<array>`, `<forward_list>`, `<unordered_map>` and `<unordered_set>` were all introduced in C++11.
`<cctype>`	Contains function prototypes for functions that test characters for certain properties (such as whether the character is a digit or a punctuation), and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa. These topics are discussed in Chapter 22, Bits, Characters, C Strings and `structs`.
`<cstring>`	Contains function prototypes for C-style string-processing functions.
`<typeinfo>`	Contains classes for runtime type identification (determining data types at execution time). This header is discussed in Section 12.9.

Fig. 6.5 | C++ Standard Library headers. (Part 2 of 4.)

Standard Library header	Explanation
`<exception>`, `<stdexcept>`	These headers contain classes that are used for exception handling (discussed in Chapter 17, Exception Handling: A Deeper Look).
`<memory>`	Contains classes and functions used by the C++ Standard Library to allocate memory to the C++ Standard Library containers. This header is used in Chapter 17, Exception Handling: A Deeper Look.
`<fstream>`	Contains function prototypes for functions that perform input from and output to files on disk (discussed in Chapter 14, File Processing).
`<string>`	Contains the definition of class `string` from the C++ Standard Library (discussed in Chapter 21, Class `string` and String Stream Processing).
`<sstream>`	Contains function prototypes for functions that perform input from strings in memory and output to strings in memory (discussed in Chapter 21, Class `string` and String Stream Processing).
`<functional>`	Contains classes and functions used by C++ Standard Library algorithms. This header is used in Chapter 16.
`<iterator>`	Contains classes for accessing data in the C++ Standard Library containers. This header is used in Chapter 15.

Fig. 6.5 | C++ Standard Library headers. (Part 3 of 4.)

Standard Library header	Explanation
`<algorithm>`	Contains functions for manipulating data in C++ Standard Library containers. This header is used in Chapter 15.
`<cassert>`	Contains macros for adding diagnostics that aid program debugging. This header is used in Appendix E, Preprocessor.
`<cfloat>`	Contains the floating-point size limits of the system.
`<climits>`	Contains the integral size limits of the system.
`<cstdio>`	Contains function prototypes for the C-style standard input/output library functions.
`<locale>`	Contains classes and functions normally used by stream processing to process data in the natural form for different languages (e.g., monetary formats, sorting strings, character presentation, etc.).
`<limits>`	Contains classes for defining the numerical data type limits on each computer platform—this is C++'s version of `<climits>` and `<cfloat>`.
`<utility>`	Contains classes and functions that are used by many C++ Standard Library headers.

Fig. 6.5 | C++ Standard Library headers. (Part 4 of 4.)