Chapter 4
Functions

Yu-Hsiang Cheng
鄭宇翔 (Slighten)
GOTC Instructor
http://m101.nthu.edu.tw/~s101021120/Myweb/index_ch.html

Recall

Variables (變數)

- 讓電腦記住資料 (data)
- 有型別、有名稱、佔記憶體空間

Operators (運算子)

- 對 data (變數、常數) 的操作 (manipulation)
- 有功能、有優先次序 (precedence)、順序關聯性 (associativity)
- Type-specific -- 不同型別的 data，操作會不同
- 事實上是一個 function -- 可自行定義自己的運算子 (taught in OOP)

Control Structures (控制結構)

- 讓電腦能做「選擇」與「重複」
- 使程式結構化 (structured)
- Conditional (Selection) -- `if`, `else`, `switch`
- Iterative (Repetition) -- `while`, `for`, `do-while`

Today's Topic

Function (函式)

- 讓你可以「重複使用」某段程式碼
- 將細節(運作方式)包裝在 sub functions 裡，使用者只需知道其輸入、輸出，不需管內部怎麼做 (API, Application Programming Interface)
  - ✓ main function 只需不斷重複使用這些 sub-functions
- 一個程式 (program) 的功能:

  讀進輸入 (read input) → 內部運作 (process) → 產生輸出 (gen. output)
- 一個函式 (function) 的功能:

  吃進參數 (pass arguments) → 內部運作 (process) → 回傳結果 (return result)
  - ✓ 一個程式就是一個大函式
  - ✓ 回憶一個程式從 main function 開始、main function 結束
- 增加可讀性 (readability)、可重複性 (reusability)

Function (函式、副函式)

- Functions 在其他語言中，也被稱作 procedures, subroutines, methods (OOP), modules (HDL)…
- A callable subprogram
- Provides abstraction (抽象化)
  - 隱藏 low-level 的細節
  - 給程式 high-level 的結構，讓 programmer 容易了解整個程式的流程 (flow)
  - Enables separable, independent development
    - ✓function 錯了修改 function 內容就好
- 分主函式(main function)、副函式(sub functions)

C++ functions

- 傳入零個或多個 arguments，回傳零個或一個 result
  - 函數是多對一
- 回傳值 (return value) 有型別 (type)

Example of High-Level Structure

```
#include <iostream>
using namespace std;

void PrintBanner(); // function declaration

int main() {
    PrintBanner();      // function call
    cout << "A simple C++ program\n";
    PrintBanner();      // function call
}

/* function definition */
void PrintBanner() {
    cout << "====================\n";
}
```

Functions in C++

- Function declaration, function prototype(函式宣告、函式原型)
  - 描述函式之 Signature (特徵) – 函式名稱、參數與其型別、回傳型別 (參數名稱可省略)
  - A **function signature** (or *type* signature, or *method* signature) defines input and output of functions or methods.

```
int factorial(int n);
```

回傳值之型別　函式名稱　參數型別　參數名稱

- Function call (函式呼叫) -- 在 expression 中使用

```
a = x + factorial(f + g);
```

1. 計算參數
2. 執行(進入→運行→回傳)函式
3. 在 expression 中使用回傳值，做運算，產生 expression 的值

Function Definition(函式定義)

- 描述 return type, name, types of arguments
  - 給每個參數一個名稱 (不一定要與 declaration 一致)
  - 其他需與 function declaration 一致

```
int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++)
        result *= i;
    return result;
}
```

Why Declaration?

Question: 函式定義已包含回傳型別與參數型別，那何必宣告？

1. 可能在看到之前先遇到 function call (使用函式)
   - 編譯器需要知道回傳型別與參數型別與數量(特徵)
2. 定義可能在別的檔案，由不同 programmers 所撰寫
   - include 一個只有 function declaration 的 "header" file
   - 分開編譯，然後 link 在一起

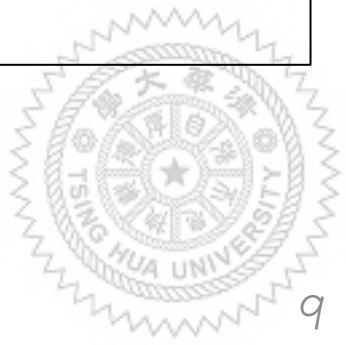
Example

```
double valueInDollars(double amount, double rate);

int main() {
    ...
    dollars = valueInDollars(francs, DOLLARS_PER_FRANC);
    cout << francs << "francs equals" << dollars << "dollars.\n";
    ...
}

double valueInDollars(double amount, double rate) {
    return amount * rate;
}
```

declaration

function call (invocation)

definition



不必宣告的方法

- 在 call 之前先 definition (把 main function 寫在最下面)

```
#include <iostream>
using namespace std;

void PrintBanner();

int main() {
    PrintBanner();
    cout << "C++\n";
    PrintBanner();
}

void PrintBanner() {
    cout << "===\n";
}
```

=

```
#include <iostream>
using namespace std;

void PrintBanner() {
    cout << "===\n";
}

int main() {
    PrintBanner();
    cout << "C++\n";
    PrintBanner();
}
```

Naming Convention (命名慣例)

- **Macro Definition:** ALL_UPPER_CASE (全大寫)

```
#define BOX_SIZE 12
```

- **Variables:**

1. **const:** ALL_UPPER_CASE

```
const int ROUNDS = 10;
```

2. **normal: 以下兩種二選一**

① all_lower_case (全小寫)，底線 (underscore) 分隔

```
int x_spacing, y_spacing, windowsize;
```

② lowerCamelCase (小駝峰式命名法)

```
int xSpacing, ySpacing, windowSize;
```

3. **flag:** is 開頭，lowerCamelCase

```
bool isEmptyString, isDone, isAlive;
```

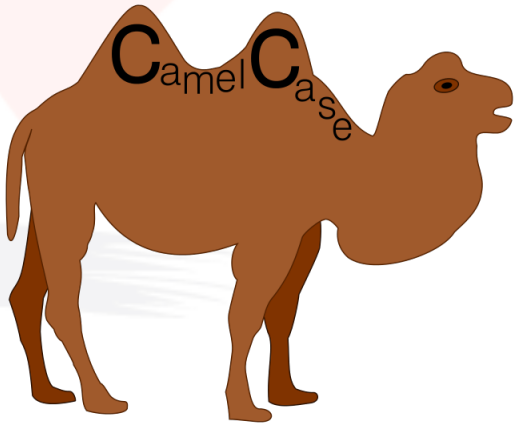
A Camel

Camel Case
(駝峰式大小寫)

UpperCamelCase (大駝峰式命名法)

- used in C++ class name (OOP)

```
class HugeInteger{
      …
}
```



lowerCamelCase (小駝峰式命名法)

- used in C++ variable name, function name

```
void readFile();
void findNearbyEnemies();
void attack(char* attacker, char* victim);
int calScore(int student_id);
```

Naming Convention (命名慣例) conti.

- 通常，變數名稱是「名詞」(東西)，函式名稱是「動詞」(做動作)
- 重點是整個程式的命名慣例要一致 (consistent)

Coding Convention (編程慣例) (1/2)

- 運算子左右各隔一個空格

```
int age = 18;
cout << "Coding convention\n";
```

- 逗號前不空格，後隔一個空格

```
attack("Slighten", "CSW");
```

- 分號前不空格，後隔一個空格

```
for (int i = 0; i < count; i++)
```

- 右括號前、左括號後不空格
- function name 接括號不空格，關鍵字接括號要空格
- 不同階層要有 indent (縮排) (按 tab 鍵)，通常是 2 或 4 個空白寬

```
if (isDead)
    cout << "Game Over!!\n";
```

Coding Convention (編程慣例) (2/2)

- 寫code請用等寬字體 (monospace font)
  - 內建：Consolas (MS), Monaco (Mac), Courier New, Andale Mono (Mac), …
  - 下載：Inconsolata-g, …
- 特色

1. 每個字等寬
2. 分的出難分的字
   1) 0, O; I, l, 1 (Consolas)
   2) 0, O; I, l, 1 (Monaco)
   3) 0, O; I, l, 1 (Andale Mono)
   - 分不出的例子:
     1. 0, O; I, l, 1 (Arial)
     2. 0, O; I, l, 1 (新細明體)

- 等寬字排名：http://www.slant.co/topics/67/~what-are-the-best-programming-fonts

English Font Family

- 5 Types

1. Serif (襯線): Times, Times New Roman, Georgia, etc.
   - 常用於新聞印刷品，襯線指的是字母端點處的裝飾用鉤角
   - 古典優雅
2. San-serif (無襯線): Verdana, **Arial Black,** Trebuchet MS, Arial, Open Sans, etc.
   - 螢幕上易讀
   - 乾淨易讀
3. Monospace (等寬): Courier New, Consolas, etc.
   - 常用於呈現程式碼
   - 像打字機
4. Cursive (行草): Comic Sans, etc.
   - 常用於標題
   - 手寫字體
5. **Fantasy** (科幻): **Impact,** Jokerman, etc.
   - 風格十足
   - 玩樂感、特殊感、科幻感

Formal vs. Actual Parameters (形式參數 vs. 真實參數)

Formal parameter: 被呼叫者 (callee) 定義的

```
double valueInDollars(double amount, double rate)
```

- 形式參數
  - amount
  - rate
- 形式參數就像 function 的 local 變數，進入函式一開始得到真實參數，當函式 return 時被消滅 (destroyed)

Actual parameters: 呼叫者 (caller) 傳入的

```
dollars = valueInDollars(23.0, 30.4);
```

- 真實參數
  - 23.0
  - 30.4
- 算完 expression 的值後，真實參數傳值 (passed by value) 給形式參數

Passing Expressions As (Actual) Parameters

- 可將 expression 傳入函式

```
int c = add(2 + x, 3);
double dollars = valueInDollars(23 + y, (45 + z)/3.2);
```

- expression 包含 function call，所以也可傳 function call

```
int c = add(2 + x, add(2, 3));
double dollars = valueInDollars(23 + valueInDollars(valueInDollars(45, 67), 78), 90);
```

- 計算順序
  - 左到右
  - 內往外

Passing (Actual) Parameters

in <cmath>

```
int myPower(int a, int b){
    int res = 1;
    for (int i = 0; i < abs(b); i++)
        res *= a;
    return res;
}

int main(){
    int x = 3, y = 5;
    cout << myPower(x, y) << endl;
    cout << myPower(x + 2, y - 1);
}
```

```
int myPower(int a, int b){
    a = x; // x in main
    b = y; // y in main

    int res = 1;
    …
    return res;
}
```

```
int myPower(int a, int b){
    a = x + 2; // x in main
    b = y - 1; // y in main

    int res = 1;
    …
    return res;
}
```

Type Conversion During Parameter Passing

- 真實參數的型別應與形式參數的型別相同
- 若不同
  1. 若是 convertible，則會自動做 type conversion (type-casting)，規則與變數規則時相同
     - e.g. char ↔ int ↔ $float$
     - `float` to `int`: truncation (3.14 → 3) (`int pi = 3.14;`)
       - `int square(int a);  square(3.14);  // 3.14 -> 3`
     - `int` to `float`: promotion (3 → 3.0) (`float t = 3;`)
       - `sqrt(3); // 3 -> 3.0`
  2. 若否，則會 compiler error

main() itself is a function!

- main() 是一個被 "loader" 呼叫的特殊函式
- 事實上， main() 的 type signature 為:

```
int main(int argc, char **argv)
```

- 回傳(給系統)一個整數
  - 0 = 正確執行
  - 非 0 = error，該值為錯誤代碼
- 參數 argc 代表參數的數量
- 參數 argv 存放所有參數
  - 想成一個字串的陣列(陣列的每個元素都放一個字串)
- void main() 在 C 可以，在 C++ 會 compiler error

Showing the arguments passed to main() (1/6)

```
int main(int argc, char **argv){
    cout << "argc = " << argc << endl;
    for (int i = 0; i < argc; i++)
        cout << "argv[" << i << "] = " << argv[i] << endl;
}
```

- 在 cmd line 上執行以下指令
- % ./a.out hello world how are you
- 輸出為：

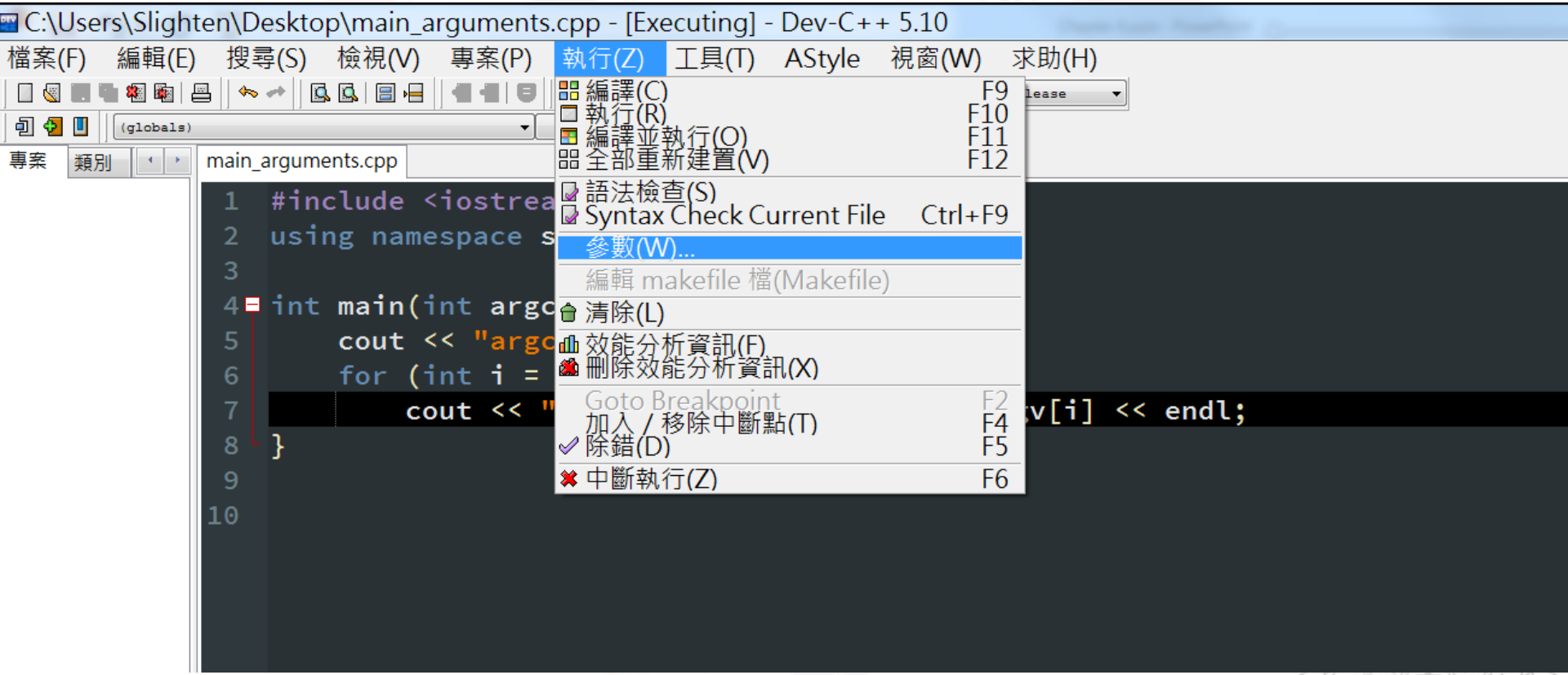
```
argc = 6
argv[0] = "./a.out"
argv[1] = "hello"
argv[2] = "world"
argv[3] = "how"
argv[4] = "are"
argv[5] = "you"
```

```
% is a prompt
./ = 當下資料夾
a.out = 執行檔檔名
```

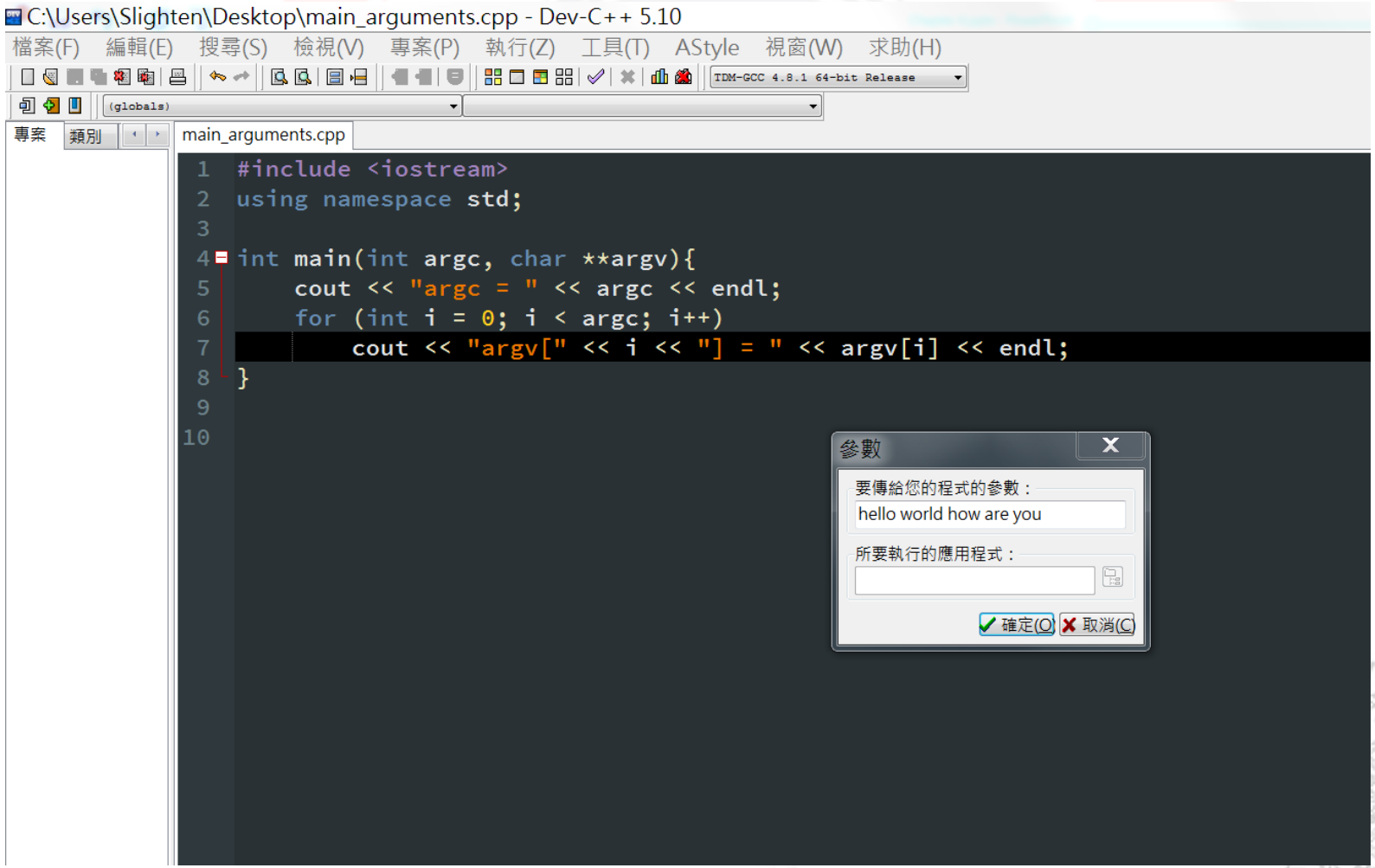
Showing the arguments passed to main() (2/6)

- 在 Dev-C++ 上的
- 執行→參數
- 輸入 `hello world how are you`
- 再編譯並執行

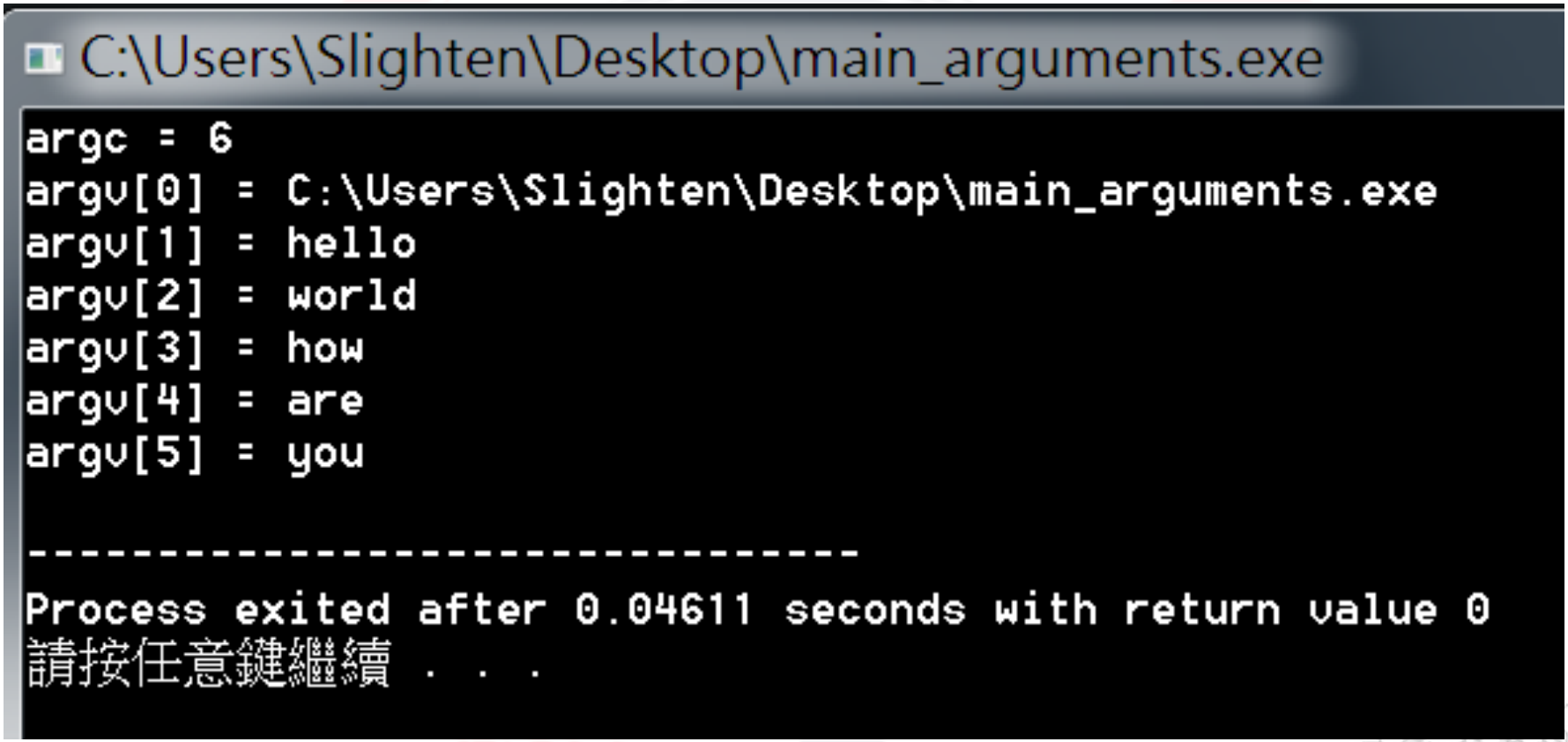
Showing the arguments passed to main() (3/6)



Showing the arguments passed to main() (4/6)



Showing the arguments passed to main() (5/6)



Showing the arguments passed to main() (6/6)

- 所以
- 在 linux 上執行編譯指令
- % g++ abc.cpp -o abc.exe
- 對 g++ 這個程式的 main() 來說：

```
argc = 4
argv[0] = "g++"
argv[1] = "abc.cpp"
argv[2] = "-o"
argv[3] = "abc.exe"
```

- 結論：我們利用以上方式，讓程式可以「吃參數」。
- 這個技巧非常常用

Some Simple Practices (1/2)

- 簡單的取絕對值程式

```
int abs(int a){
    if (a < 0)
        return -a;
    else
        return a;
}
```

- 利用？：運算子達到更簡潔的效果

```
int abs(int a){
    return a < 0 ? -a : a;
}
```

Some Simple Practices (2/2)

- Question: 以下程式碼會怎樣？

```
int abs(int a){
   if (a < 0)
      return -a;
   return a;
}
```

- 答案是：效果一樣，因為一碰到 return 就回去呼叫者那邊了。

Recall

- 回想一下
- 區域變數與全域變數
- 能見度與生存時間

local_var 的能見度與生存時間

global_var 的能見度與生存時間

local_var 的能見度與生存時間

abc.cpp

```cpp
#include <iostream>
using namespace std;

int global_var = 7; // 全域變數

void foo(){
    int local_var = 5; // 區域變數
    /* 這裡的 local_var 與
     * main function 裡的 local_var 不一樣*/
    cout << local_var + global_var << endl;
}

int main(int argc, char **argv){
    int local_var = 3; // 區域變數
    /* 這裡的 local_var 與
     * function foo() 裡的 local_var 不一樣 */
    cout << local_var + global_var << endl;
    return 0;
}

```

Question I

- 哪個 cout 先？經過 foo() 後， main() 裡的變數 a 的值有變化嗎？
- 以下程式的輸出是？

```
int foo(int a){
    cout << a << '\n';
    a = 1;
    return a;
}

int main(){
    int a = -5, b;
    b = foo(a);
    cout << a << ' ' << b << endl;
}
```

int a = a; // 想像先做

- 提示：main() 裡的變數 a 與 foo() 裡的參數 a 的關係是？
- 答：完全是 2 個不同的變數。

Question II (1/2)

- foo() 改不到 main() 裡變數 a 的值
- 若想在 foo() 裡改 main()的 變數 a 的值怎麼做？

1. 傳遞 a 的記憶體位置 (利用指標變數儲存，Ch6)

```
int a = a;      // 想像先做
int foo(int a){
    cout << a << '\n';
    a = 1;
    return a;
}

int main(){
    int a = -5, b;
    b = foo(a);
    cout << a << ' ' << b
         << endl;
}
```

```
int* a = &a;      // 想像先做
int foo(int* a){
    cout << *a << '\n';
    *a = 1;
    return *a;
}

int main(){
    int a = -5, b;
    b = foo(&a);
    cout << a << ' ' << b
         << endl;
}
```

a (指標變數)存放記憶體位置

用 * 運算子找到 a 所存的值 (現在存 main()的 a 的位置) 底下，去取值

用 & 運算子取出 a 所在的記憶體位置

Question II (2/2)

2. 傳遞 a 的參考 (C++'s feature)

跟 main() 傳進去的 a 是同一個！

```
int foo(int a){int a = a;      // 想像先做
    cout << a << '\n';
    a = 1;
    return a;
}

int main(){
    int a = -5, b;
    b = foo(a);
    cout << a << ' ' << b << endl;
}
```

```
int foo(int &a){  int &a = a; // 想像先做
    cout << a << '\n';
    a = 1;
    return a;
}

int main(){
    int a = -5, b;
    b = foo(a);
    cout << a << ' ' << b << endl;
}
```

傳值 vs. 傳指標 vs. 傳參考

- a 是真實參數，b 是形式參數
- `int a = 3;`
- 傳值：`int b = a;`
  - 像複製貼上
- 傳指標：`int *b = &a;`
  - `&a` 取出變數 a 所在位置，給指標變數 b 存
- 傳參考：`int &b = a;`
  - b 是 a 的「別名」(alias)

傳遞方式	空間使用	執行效率	資料安全	語法容易度
數值	劣	劣	優	優
指標	優	優	劣	劣
參考	優	優	劣	優

*a, &a, int *a, int &a?

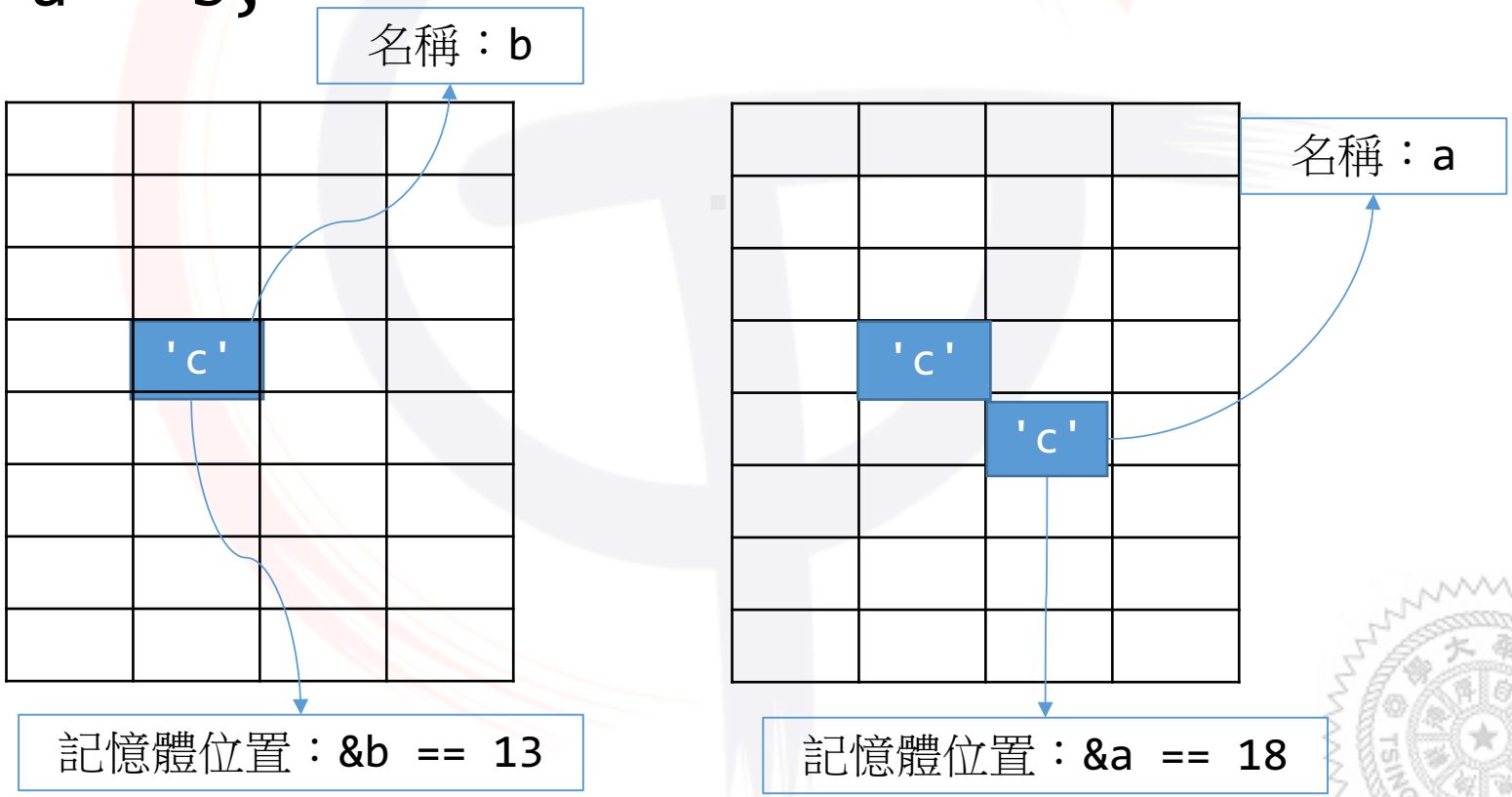
- `int *a;` 是定義 a 為指標變數，存一個記憶體位置(地址)，該地址住著一個整數
- *a 是尋著 a 存放的值(地址) 去該地址找人 (取值)，又稱解參考(dereference)
- `int &a;` 是定義 a 為參考變數，成為別人的參考(別名)
- &a 是取出變數 a 所在的記憶體位置(地址)，又稱參考(reference)

Example

```
int a = 3;
cout << &a << endl;       // assume 0x22fe44
cout << *a << endl;       // illegal

int *b = &a;              // b points to a, b = 0x22fe44, same
as {int *b; b = &a;}
cout << b << endl;        // == 0x22fe44
cout << *b << endl;       // == 3
*b = 5;                   // a gets 5
cout << &b << endl;       // assume 0x22fe38

int &c = a;               // c is a's alias
cout << c << endl;        // = 5
c = 1;                    // a gets 1
cout << &c << endl;       // == 0x22fe44
cout << *c << endl;       // illegal
```

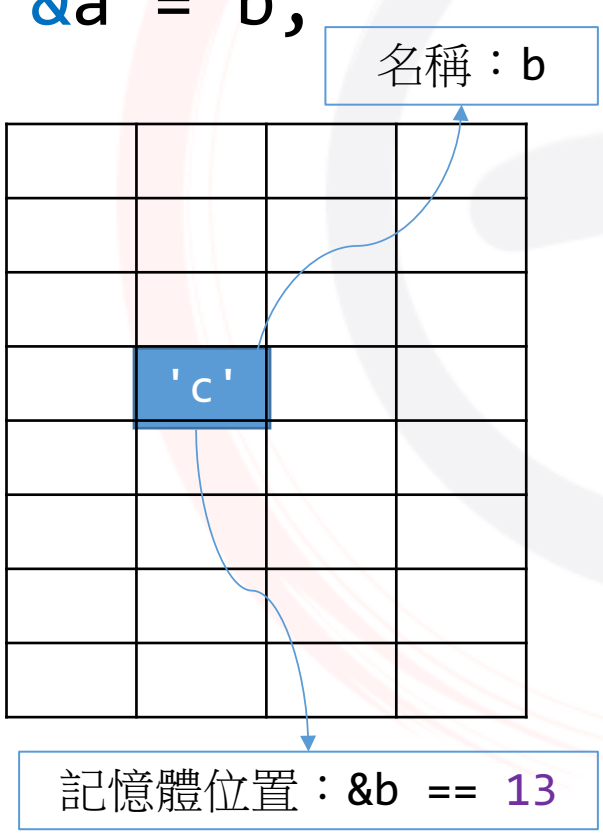
圖示 (傳值)

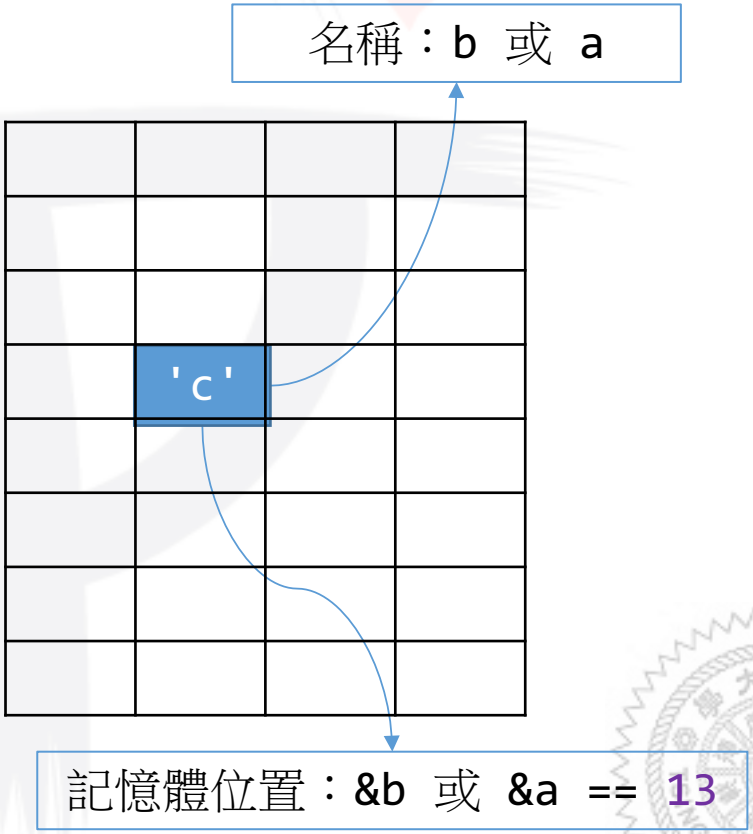
```
char b = 'c';
char a = b;
```



圖示 (傳參考)

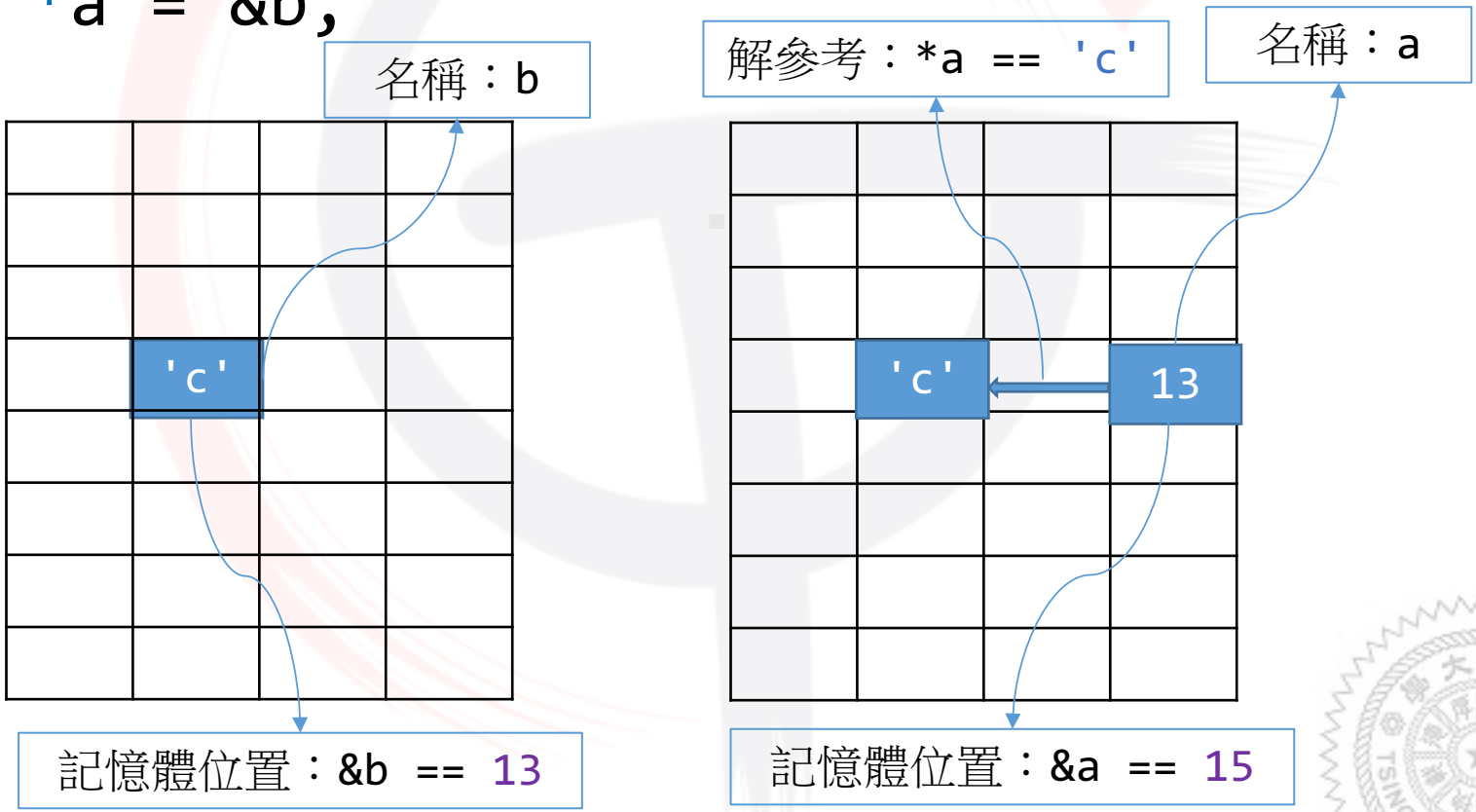
```
char b = 'c';
char &a = b;
```





圖示 (傳指標)

```
char b = 'c';
char *a = &b;
```



Problem 1: Case Conversion

- Convert lower case to upper case

```
#include <iostream>
using namespace std;
char toUpper(char inchar);

int main() {
    char echo, upcase;
    while (cin >> echo) {
        upcase = toUpper(echo);
        cout << upcase;
    }
}

char toUpper(char inchar) {
    if ('a' <= inchar && inchar <= 'z')
        return inchar - ('a' - 'A');
    return inchar;
}
```

exit by typing Ctrl-Z or Ctrl-D

note that echo NEVER gets changed

C Library

- `#include <stdio.h> /* standard I/O */`
- `#include <math.h> /* math library */`
- `#include <ctype.h> /* character types */`
- `#include <time.h> /* related to time, date, etc */`
- `#include <string.h> /* related to string manipulation, 跟 C++ 的 <string> 不一樣 */`
- `#include <stdlib.h> /* standard library: string <-> number conversion, memory allocation, command execution, random number generator ..*/`
- `#include <curses.h> /* CRT screen handling */`
- `#include <sys/socket.h> /* communication */`
- `#include <sys/types.h>`
- …

Example: Math Library

- #include <cmath> // or <math.h>
- sqrt(x) = square-root
- exp(x) = exponentiation
- log(x) = natural log of x
- log10(x) = base-10 log of x
- fabs(x) = floating-point absolute value of x
- ceil(x) = "ceiling" of x (e.g., ceil(9.2) is 10.0)
- floor(x) = "floor" of x
- pow(x, y) = x raised to the power of y (x 的 y 次方)
- fmod(x, y) = floating point modulo (x % y)
- sin(x) = sine of x (x in radians, not degrees. pi = 180 degrees)
- cos(x) = cosine of x
- tan(x) = tangent of x

Using library functions

```
#include <iostream>
#include <cmath>
using namespace std;

double square(double y) {
    return pow(y, 2);
}

int main() {
    double x;
    cin >> x;
    cout << square(x);
    return 0;
}
```

這個 pow(x, y) 來自於 <cmath>

如果宣告 int 且輸入 2.5 時，答案會是？

In Linux, Compile this with
% g++ main.cpp -lm
-l means link with, and
m means math library

Making your own library (1/2)

- 將 code 從 main.cpp 分開
- 創 .h 和對應的 .cpp 檔
- Header file: sq.h （放函式宣告、原型）

```
#ifndef __SQ_H__
#define __SQ_H__
double square(double y); /* function prototype */
#endif __SQ_H__
```

- C++ source file: sq.cpp （放函式定義、實作）

```
#include <cmath>
double square(double y) {
    return pow(y, 2);
}
```

Making your own library (2/2)

- #include "sq.h" 使得能在 main.cpp 裡使用 square(x)

```
#include <iostream>
#include "sq.h" /* this is my own header file! Use "" not <> */
using namespace std;

int main() {
    double x;
    cin >> x;
    cout << square(x);
    return 0;
}
```

- Conclusion
  - main.cpp: 主函式
  - sq.h: square() 的自訂頭檔(函式宣告)
  - sq.cpp: square() 的實作

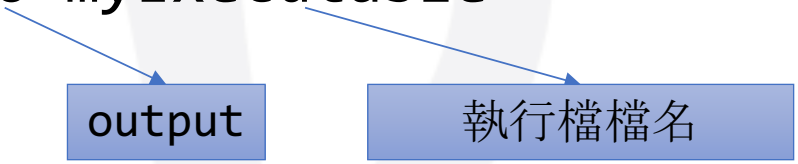
Compile files separately and then link (CLI)

1. 分開編譯

```
% g++ -c sq.cpp # this creates the object file sq.o
% g++ -c main.cpp # this creates the object file main.o
```

2. 然後 link

```
% g++ sq.o main.o -o myExecutable
```



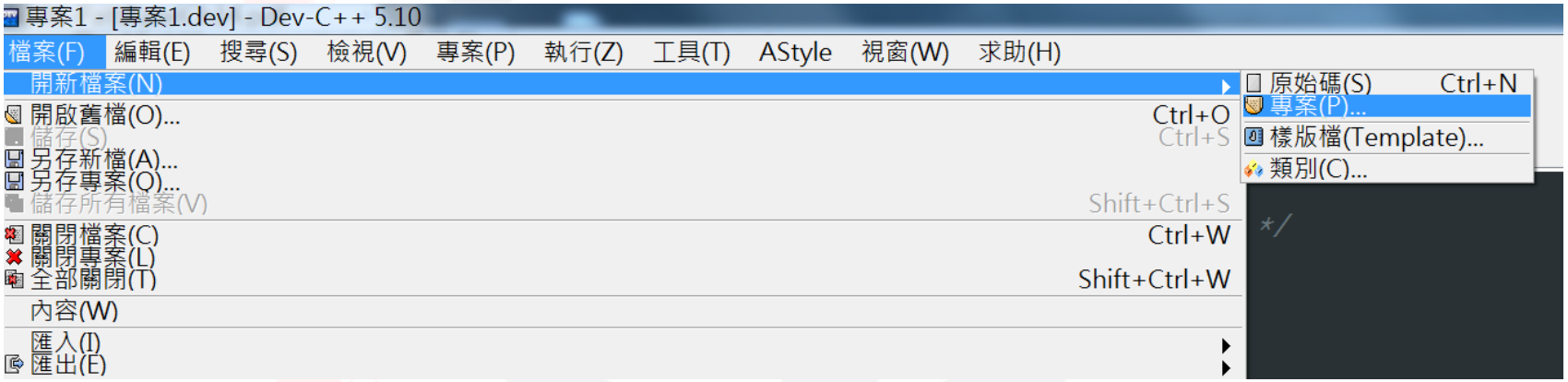
3. 執行

```
% ./myExecutable
```

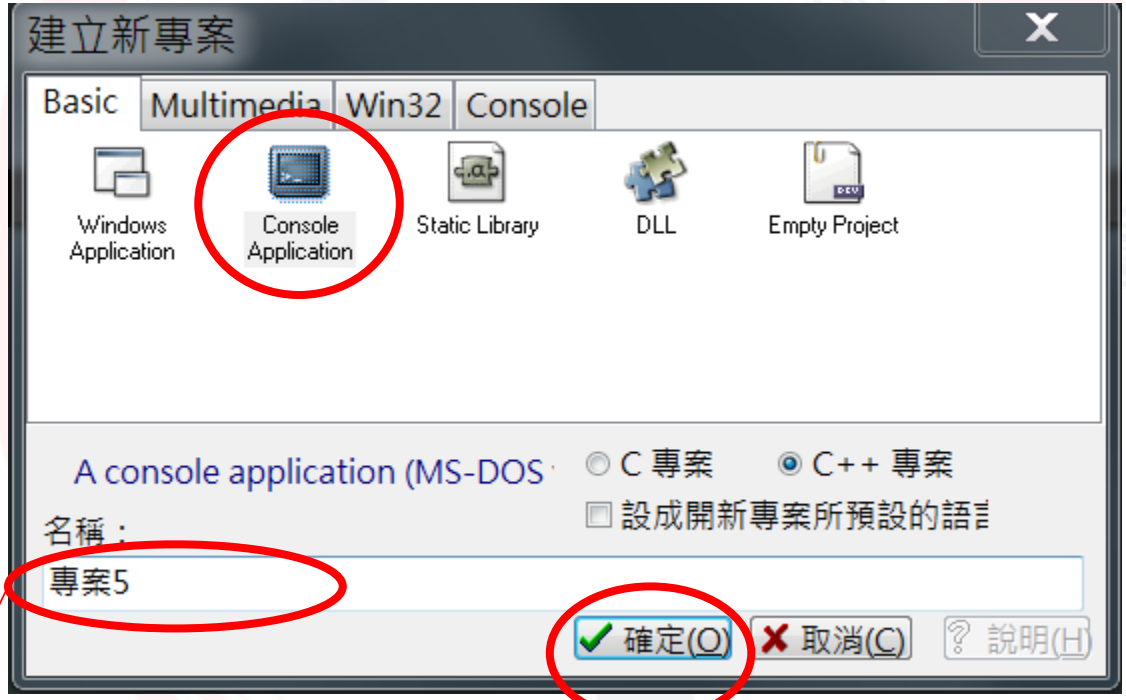
- 你可以只給別人 .o 跟 .h 檔，不必給別人 .cpp 檔 (智慧財產)
- 別人拿到你的 .o 跟 .h 即可 link 並使用裡面的 functions

Compile files separately and then link (Dev-C++) (1/3)

- 檔案→開新檔案→專案



Compile files separately and then link (Dev-C++) (2/3)



隨意

Compile files separately and then link (Dev-C++) (3/3)

- 新增原始碼，加到專案裡，存成 sq.h 與 sq.cpp
- 編輯 main.cpp
- 編譯並執行

細談變數能見度 (Scoping) 與記憶體空間

- 4 種 scopes

1. Global, 跨檔案
2. Global, 只在一個檔案內
3. Local, 在一個 function 內
4. Local, 在一個 function 的 { } 內

- 2 種存放的記憶體空間

1. Heap (global, static) 堆積
2. Stack (auto local, nested) 堆疊

Scoping of Identifiers in C++ (1/2)

1. **Global, 跨檔案**
   - 變數：在所有 function 外宣告，在另個檔案加上關鍵字 extern
     - `int i; /* in main.cpp, outside a function -- only one file may declare so */`
     - `(in .h file) extern int i; /* use i in main.cpp */`
   - 函式： declare it normally
2. **Global, 只在一個檔案內**
   - 在 global 宣告加上關鍵字 static
     - `static int i; /* outside a function -- visible in this file only! */`
     - `static int myFunction(int a, int b); /* also visible within file only */`
   - 別的檔案可宣告相同名字的 global 變數或函式

- 通常，盡量不要用 global 變數，要的話就 static 它。

Scoping of Identifiers in C++ (2/2)

3. Local, 在一個 function 內

- 變數：在 function 內宣告
  - `int i; /* inside a function -- visible only inside the function */`
- 函式：C++ 不能在函式裡定義一個新函式！！

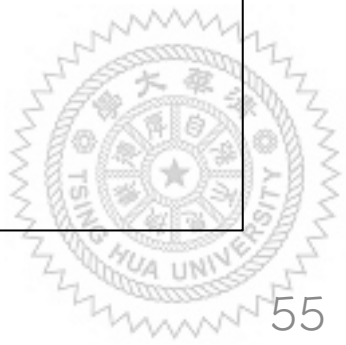
4. Local to a { } block

- 在巢狀 { } 內，內部名稱屏蔽(shadow, mask)外部名稱。

```
int x = 5; /* x is global */
int main(int argc, char **argv) { /* main is global */
    char x = 'a'; /* this local x masks the global x */
    if (argc) {
        double x = 3.14; /* this inner x masks the main's x and global x */
        cout << "x = " << x << "global x = " << ::x << endl;
        float main; /* this main masks the function's own name! */
    }
}
```

Example of Identifier Masking

```
/* buggy function. what's wrong? */
int maximum(int x, int y, int z) {
   int max = x;
   if (y > max) {
      int max = y;
   }
   if (z > max) {
      max = z;
   }
   return max;
}
float g;
```



Two kinds of static

- Static local (vs. auto local)
  - local 變數 + `static`
    => 存放在 heap (與 global 同)但可見度是 local
  - Example: `i` (on line 2)
  - 一般變數叫作 auto local (存放在 stack)
  - 目的： 在多次呼叫間記住 state

- Static global
  - 與 p.50 – 2 同

```
static int nextNumber() {
    static int i = 0;
    return i++;
}
static float g;
```

Some Practices

作業I: 簡單月曆 part 1

- 檔名叫做 cal_part1.cpp
- 此 part 要各位先印出月曆的雛形

1. prompt "enter the starting day of week: 0 = Sunday, 1 = Monday, etc: "
2. prompt "enter the number of days in the month: "
3. 印出月曆

Sample Output I

```
enter the starting day of week: 0 = Sunday, 1 = Monday, etc: 0
enter the number of days in the month: 31
Sun Mon Tue Wed Thu Fri Sat
  1   2   3   4   5   6   7
  8   9  10  11  12  13  14
 15  16  17  18  19  20  21
 22  23  24  25  26  27  28
 29  30  31

--------------------------------
Process exited after 12.66 seconds with return value 10
請按任意鍵繼續 . . .
```

Sample Output II

```
enter the starting day of week: 0 = Sunday, 1 = Monday, etc: 5
enter the number of days in the month: 29
Sun Mon Tue Wed Thu Fri Sat
                        1   2
  3   4   5   6   7   8   9
 10  11  12  13  14  15  16
 17  18  19  20  21  22  23
 24  25  26  27  28  29

--------------------------------
Process exited after 2.321 seconds with return value 10
請按任意鍵繼續 . . .
```

Sample Output III

```
enter the starting day of week: 0 = Sunday, 1 = Monday, etc: 13
enter the number of days in the month: 50
Sun Mon Tue Wed Thu Fri Sat
                        1
  2   3   4   5   6   7   8
  9  10  11  12  13  14  15
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30  31  32  33  34  35  36
 37  38  39  40  41  42  43
 44  45  46  47  48  49  50


--------------------------------
Process exited after 10.81 seconds with return value 10
請按任意鍵繼續 . . .
```

Hint

1. 用 setw() 讓數字靠右 (include <iomanip>)
2. 如何算印的空格數目？
3. 先處理起始日 0~6，天數 28~31 的例子
4. 起始日超過 6， 如何變成 0~6 (回想 rand() 的例子)
5. 何時換行？
6. 可能用到 for loop
7. 可能用到 %
8. 可能用到 ? : 運算子

作業II: 簡單月曆 part 2

- 檔名叫做 cal_part2.cpp
- 此 part 要各位做出月曆的核心(算出起始日和該月天數)

1. prompt "enter the year (1753–2099): "
2. prompt "enter the month (1–12): "
3. print "The month has ? days"
4. print "The first day of this month is a ?"

Sample Output I

```
enter the year (1753-2099): 1753
enter the month (1-12): 1
The month has 31 days
The first day of this month is a Monday

--------------------------------
Process exited after 15.48 seconds with return value 10
請按任意鍵繼續 . . .
```

Sample Output II

```
enter the year (1753-2099): 1754
enter the month (1-12): 1
The month has 31 days
The first day of this month is a Tuesday

--------------------------------
Process exited after 3.264 seconds with return value 10
請按任意鍵繼續 . . .
```

Sample Output III

```
enter the year (1753-2099): 2015
enter the month (1-12): 8
The month has 31 days
The first day of this month is a Saturday

--------------------------------
Process exited after 1.888 seconds with return value 10
請按任意鍵繼續 . . .
```

Hint

1. 已知西元 1753 年 1 月 1 日是星期一
2. 先運用之前寫過的閏年判斷程式碼來判斷閏年
3. 再依月份來知道該月有幾天 (28 ~ 31)
4. 接著就能知道該年有幾天 (365 or 366)
5. 再算出現在輸入的年份的 1 月 1 日與 1753 年 1 月 1 日相隔幾天(for-loop)
6. 再算出現在輸入的月份的 1 日與該年的 1 月 1 日隔幾天(for-loop)
7. 取餘數，就能知道該年該月的 1 日是星期幾
8. 在 linux 上利用 **cal [月] [年]** 來得知該月月曆
   - e.g. % cal 1 1973

作業III: 簡單月曆 part 3

- 檔名叫做 cal_part3.cpp
- 組合 part1, part2 印出超猛月曆
- 須利用 main() 吃參數的方式

1. 無參數(./a.out)：印出執行當下年月的月曆
2. 吃 1 個參數為月分(./a.out *month*)： 印出執行當下年指定月的月曆
3. 吃 2 個參數分別為月分與年分(./a.out *month year*)： 印出指定年指定月的月曆

Sample Output I

- 無參數 (./a.out)：印出執行當下年月的月曆

```
          August 2015
Sun Mon Tue Wed Thu Fri Sat
                            1
  2   3   4   5   6   7   8
  9  10  11  12  13  14  15
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30  31
```

Sample Output II

- 吃 1 個參數 2 (./a.out 2) ： 印出執行當下年指定月的月曆

```
          February 2015
Sun Mon Tue Wed Thu Fri Sat
  1   2   3   4   5   6   7
  8   9  10  11  12  13  14
 15  16  17  18  19  20  21
 22  23  24  25  26  27  28
```

Sample Output III

- 吃 2 個參數分別是 1 2000 (./a.out 1 2000) ： 印出指定年指定月的月曆
- 比較系統指令 cal

```
Slighten@SlightenCheng ttys000-bash 09:35 ~/codes/coin $ cal 1 2000
      1月 2000
日 一 二 三 四 五 六
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

```
     January 2000
Sun Mon Tue Wed Thu Fri Sat
                          1
  2   3   4   5   6   7   8
  9  10  11  12  13  14  15
 16  17  18  19  20  21  22
 23  24  25  26  27  28  29
 30  31
```

Requirements

- 須用到以下 4 個規定的 function prototype
- `bool isLeapYear(int year);`
  - `/* returns true if the year is a leap year, false if not */`
- `int numberOfDaysInYear(int year);`
  - `/* returns the number of days (365 or 366) the year has */`
- `int startingDayOfWeekOf(int month, int year);`
  - `/* returns the starting day of the week of the month and year. 0 = Sunday, 1 = Monday, ... 6 = Saturday */`
- `void printGenericCalendar(int month, int year, int startWday, int daysInMonth);`
  - `/* prints the calendar for the month and year, whose first day starts on startWday (0 = Sunday, 1 = Monday, .. 6 = Saturday; and the month has daysInMonth days. This is just a printing routine... it does not know how many days the month has, nor does it know what day of the week first day is. */`

Hint I

1. 宣告 `int main(int argc, char **argv){ }`
2. 利用 `argc` 得知你吃到幾個參數
3. 注意 `argv[i]` 是字串而不是數字
   - 需要用 `atoi()` 函式將字串轉成數字 (`#include <cstdlib>`)
   - google `C++ atoi` 查看它的用法
   - http://www.cplusplus.com/reference/cstdlib/atoi/

Hint II

1. 如何得知系統當下年月日？（複製以下程式碼）

```
#include <ctime>

void getToday(int *month, int *day, int *year) {
    time_t result;
    struct tm *tr;
    time(&result);
    tr = localtime(&result);
    *month = tr->tm_mon + 1;
    *day = tr->tm_mday;
    *year = tr -> tm_year + 1900;
}
```

2. 在 main() 裡利用以下兩行得到當下年月日

```
int month, day, year;
getToday(&month, &day, &year);
```

作業IV: 猜數字加強版

- 檔案命名成 guess_AI.cpp
- 基本功能與 Mini Project I 一樣 (Ch3)
- 多加 mode 選擇功能，P1 vs COM or P1 vs P2
- 電腦會自己猜，但絕不會猜超出 range 的數字
- You may use functions.

Simple Flow I

```
Choose a mode (C or c: P1 vs COM, others: P1 vs p2): c
P1 enter the number (0~99): 50
Too large!
COM enter the number (0~49): 1
Too small!
P1 enter the number (2~49): 18
Too small!
COM enter the number (19~49): 40
Too large!
P1 enter the number (19~39): 25
Too large!
COM enter the number (19~24): 23
Too large!
P1 enter the number (19~22): 19
Too small!
COM enter the number (20~22): 22
Too large!
P1 enter the number (20~21): 20
P1 Win!

--------------------------------
Process exited after 28.6 seconds with return value 0
請按任意鍵繼續 . . .
```

Simple Flow II

```
Choose a mode (C or c: P1 vs COM, others: P1 vs p2): C
P1 enter the number (0~99): 28
Too small!
COM enter the number (29~99): 41
Too small!
P1 enter the number (42~99): 45
Too small!
COM enter the number (46~99): 79
Too large!
P1 enter the number (46~78): 47
Too small!
COM enter the number (48~78): 63
Too large!
P1 enter the number (48~62): 48
Too small!
COM enter the number (49~62): 59
Too large!
P1 enter the number (49~58): 55
Too small!
COM enter the number (56~58): 56
COM Win!

--------------------------------
Process exited after 14.86 seconds with return value 0
請按任意鍵繼續 . . .
```

Simple Flow III

```
Choose a mode (C or c: P1 vs COM, others: P1 vs p2): D
P1 enter the number (0~99): 58
Too large!
P2 enter the number (0~57): 50
Too large!
P1 enter the number (0~49): 30
Too large!
P2 enter the number (0~29): 15
Too small!
P1 enter the number (16~29): 25
Too small!
P2 enter the number (26~29): 28
Too small!
P1 enter the number (29~29): 29
P1 Win!

--------------------------------
Process exited after 18.92 seconds with return value 0
請按任意鍵繼續 . . .
```

Simple Flow IV

```
Choose a mode (C or c: P1 vs COM, others: P1 vs p2): C
P1 enter the number (0~99): 50
Too large!
COM enter the number (0~49): 2
Too small!
P1 enter the number (3~49): -1

--------------------------------
Process exited after 7.26 seconds with return value 1
請按任意鍵繼續 . . .
```

Template (1/2)

```
#include <iostream>
#include <cstdlib> // rand(), srand()
#include <ctime>   // time()
using namespace std;
static int lower = 0, upper = 99, guess = -2, mode;
bool turn = false; // turn = 0 and 1, mapping player 1 and 2

int chooseMode(){
    /* cin 得到的字元若是 'C' 或 'c'
     * 則 return ???
     * 否則 return ??? */

    /* ??? */
}

int whoGuess(){
    /* 是 1P 還是 2P 還是 COM (mode 跟 turn 決定)
     * 若是 COM，則亂數猜 (range?)
     * 若是 1P or 2P，則 cin */

    /* ??? */
}
```

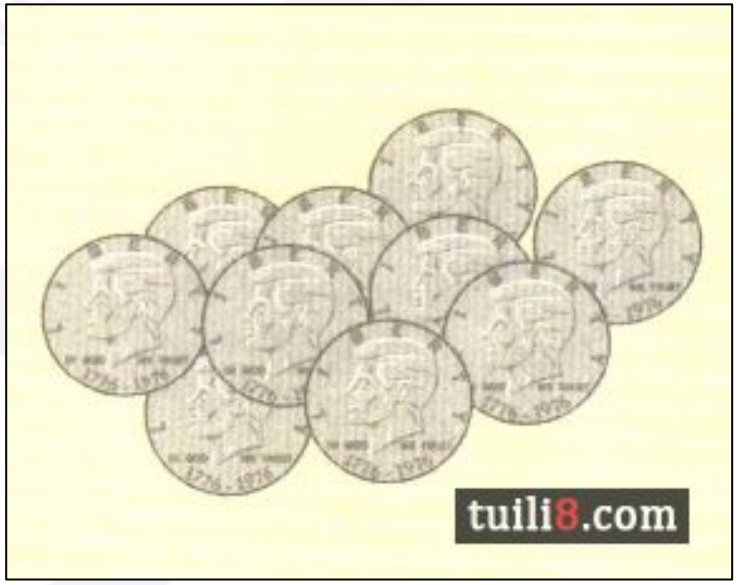
Template (2/2)

```
void winJudge(int answer){
    /* 利用 guess 與 answer 關係，判斷輸贏 */
    if (guess == -1) exit(1);
    else if (/* ??? */){
        /* ??? */
    }
    /* ??? */
}

int main(int argc, char **argv){
    int answer;
    srand(time(NULL)); // new seed
    answer = rand() % 100; // random from 0~99
    mode = chooseMode();
    while (guess != answer){
        guess = whoGuess();
        winJudge(answer);
        turn = !turn; // 1 -> 0, 0 -> 1
    }
    return 0;
}
```

Fun Project
取硬幣賽局與其推廣

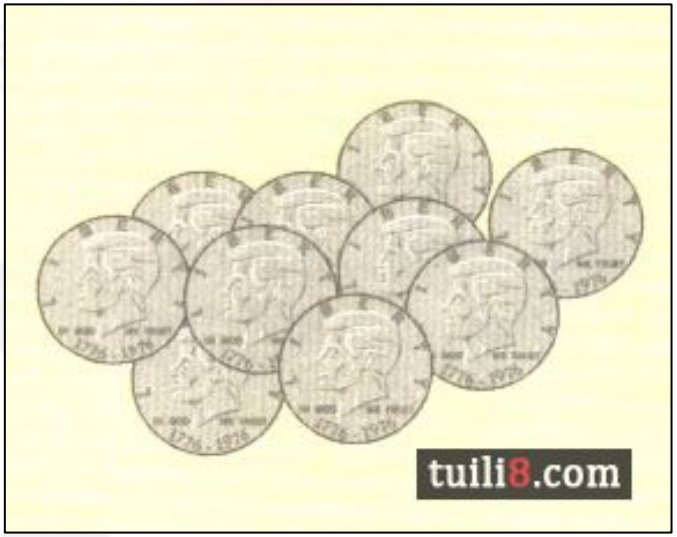
原始的取硬幣賽局

- 桌面上一開始有 10 枚硬幣
- 有 2 名玩家輪流拿走硬幣
- 1 次可以拿走 1 或 2 枚
- 拿到第 10 枚的人就輸了



試玩

- 玩家 A, B
- A 先拿走 2 枚，剩 8 枚
- B 再拿走 1 枚，剩 7 枚
- A 再拿走 1 枚，剩 6 枚
- B 再拿走 2 枚，剩 4 枚
- A 再拿走 2 枚，剩 2 枚
- B 再拿走 1 枚，剩 1 枚，A 輸了



取自甲斐谷忍所著的日本漫畫《詐欺遊戲》

取自日本漫畫《詐欺遊戲》，甲斐谷忍

取自甲斐谷忍所著的日本漫畫《詐欺遊戲》

必勝法 (1/2)

- 事實上這個遊戲這個情況是後手必勝
- 當然如果你是後手你不會玩還是會輸

必勝法 (2/2)

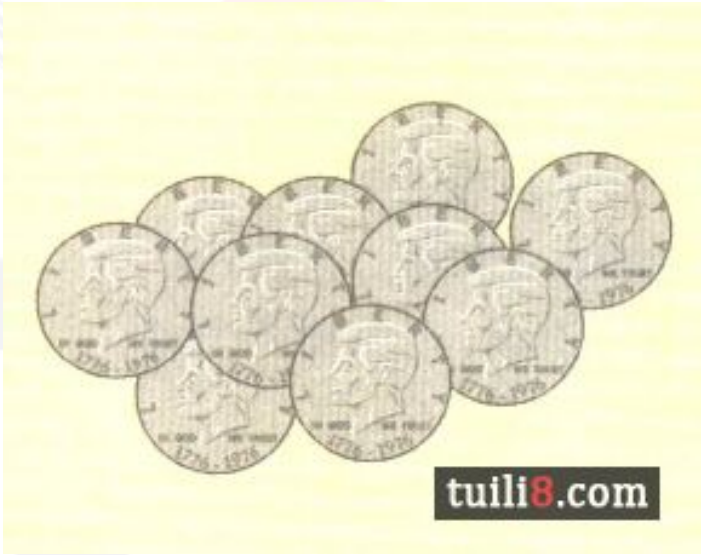
- 這個遊戲最重要的關鍵是「拿到第 10 枚的人就輸了」
- 你要讓對方輸，第 9 枚，必須由你拿到
- 再往前推，為了拿到第 9 枚，
  第 6 枚，必須由你拿到
- 再往前推，為了拿到第 6 枚，
  第 3 枚，必須由你拿到

- 因為 1 次只能拿 1 或 2 枚
- 你只要是後手，必能拿到第 3 枚，因此後手必勝

- 這種從結局推回來的方法，稱為**反向歸納法 (Backward Induction)**
- 想一想：如果是 1 次可以拿 1~3 枚，仍是後手必勝嗎？

推廣的取硬幣賽局

- 桌面上一開始有 n 枚硬幣
- 有 2 名玩家輪流拿走硬幣
- 1 次可以拿走 1 ~ m 枚
- 拿到第 n 枚的人就輸了



- 想想看必勝法是？
- 問題

1. 是先手必勝還是後手必勝
2. 要取哪幾枚才能獲勝？

- 提示：需用到 %，不需用到 Array

Fun Project

- 寫一個玩家與AI PK，AI 必勝的取硬幣賽局的程式

```
Slighten@SlightenCheng ttys000-bash 08:25 ~/codes/coin $ g++ main.cpp -o takeCoin

Slighten@SlightenCheng ttys000-bash 08:27 ~/codes/coin $ ./takeCoin
遊戲：   取硬幣賽局
玩法：   桌上有 n 枚硬幣，兩名玩家輪流拿走硬幣，
        一次只能拿走至多 m 枚硬幣，不可以不拿，
        誰拿到最後一枚硬幣即為輸家。
        現在你身為一名玩家，要對上另一位玩家「最強電腦」，
        試著打敗他吧！
聲明：   這是一個電腦必勝的程式...

桌上有幾枚硬幣？ 10
一次能取至多幾枚硬幣？ 2
遊戲開始！

----------第 1 回合----------
玩家先手：
玩家選擇拿幾枚硬幣？ 3
拿太多了！
玩家選擇拿幾枚硬幣？ 2
桌上還剩下 8 枚硬幣
換電腦選擇

----------第 2 回合----------
電腦選擇拿幾枚硬幣？ 1
桌上還剩下 7 枚硬幣
換玩家選擇

----------第 2 回合----------
電腦選擇拿幾枚硬幣？ 1
桌上還剩下 7 枚硬幣
換玩家選擇

----------第 3 回合----------
玩家選擇拿幾枚硬幣？ 2
桌上還剩下 5 枚硬幣
換電腦選擇

----------第 4 回合----------
電腦選擇拿幾枚硬幣？ 1
桌上還剩下 4 枚硬幣
換玩家選擇

----------第 5 回合----------
玩家選擇拿幾枚硬幣？ 2
桌上還剩下 2 枚硬幣
換電腦選擇

----------第 6 回合----------
電腦選擇拿幾枚硬幣？ 1
桌上還剩下 1 枚硬幣
玩家輸了！

是否重新遊戲 (y/n)？ n
```