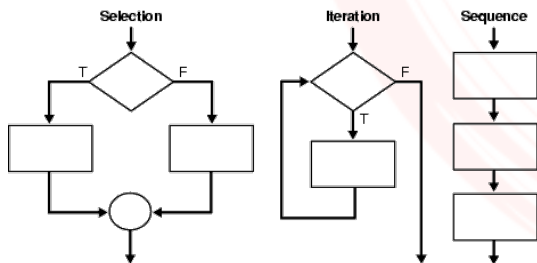


Chapter 3

Control Structures



Yu-Hsiang Cheng
鄭宇翔 (Slighten)
GOTC Instructor

http://m101.nthu.edu.tw/~s101021120/Myweb/index_ch.html



Outline

- ✓ Some Tips
- ✓ Conditional
- ✓ Iteration
- ✓ Useful Skills
- ✓ Some Practices



Outline

- ✓ Some Tips
- ✓ Conditional
- ✓ Iteration
- ✓ Useful Skills
- ✓ Some Practices



打字時或複製投影片上程式碼時請注意

- “ ” 這兩個是中文的雙引號，" 才是英文的雙引號（要用這個）
- ‘ ’ 這兩個是中文的單引號，' 才是英文的單引號（要用這個）
- ' ' 這裡面是中文全形空白，' ' 裡面才是英文的半形空白（要用這個）
- ？ ！ ， ； 「 」 （ ） % # @ & | + _ - 這些都是中文符號，要用 ？ ！ ， ； { } [] () % # @ & | + _ -



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, window_size;
```

- ② lowerCamelCase (小駝峰式命名法)

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

3. flag: is 開頭 , lowerCamelCase

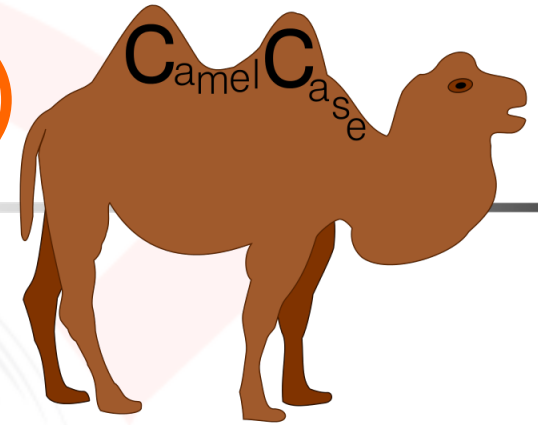
```
bool isEmptyString, isDone, isAlive;
```



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) (cont'd)

- 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)

- 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



Control Structures

Control structures 使程式能 1. 選擇 2. 重複

1. Conditional (Selection)

- 根據 expression 的值的「真(true)或假(false)」來決定接下來要執行哪段code (要走哪條路)。
- if
- if-else
- switch

2. Iteration (Repetition)

- 根據 expression 的值的「真(true)或假(false)」來決定接下來要不要重複執行這段 code。
- while
- for
- do-while



Recall (1/3)

Expression

- 變數、常數、運算子與函式呼叫的任意組合
- 本身有 type 也有 value

Statement

- 一個完整的執行單元
- Expression + 分號
- Simple 與 Compound (大括號裡很多 simple)

In C++

- 「0」是「false」
- 「非 0」是「true」



Recall (2/3)

Truth Table (A and B are expressions)

- Logical Operators -- Logical NOT, Logical AND and Logical OR (!, &&, and ||)

NOT

A	!A
T	F
F	T

AND

A	B	A && B
T	T	T
T	F	F
F	T	F
F	F	F

只要有一個是 0 即為 0
0 is the dominant term

OR

A	B	A B
T	T	T
T	F	T
F	T	T
F	F	F

只要有一個是 1 即為 1
1 is the dominant term



Recall (3/3)

- If `int x = 0;`

Expression	Type	Value	True or false	Value of x after executed
<code>x</code>	<code>int</code>	<code>0</code>	<code>false</code>	<code>0</code>
<code>x != 0</code>	<code>bool</code>	<code>false</code>	<code>false</code>	<code>0</code>
<code>!x</code>	<code>bool</code>	<code>true</code>	<code>true</code>	<code>0</code>
<code>x == 0</code>	<code>bool</code>	<code>true</code>	<code>true</code>	<code>0</code>
<code>x + 1</code>	<code>int</code>	<code>1</code>	<code>true</code>	<code>0</code>
<code>x - 1</code>	<code>int</code>	<code>-1</code>	<code>true</code>	<code>0</code>
<code>x += 1</code>	<code>int</code>	<code>1</code>	<code>true</code>	<code>1</code>
<code>x = 1</code>	<code>int</code>	<code>1</code>	<code>true</code>	<code>1</code>
<code>x++</code>	<code>int</code>	<code>0</code>	<code>false</code>	<code>1</code>
<code>++x</code>	<code>int</code>	<code>1</code>	<code>true</code>	<code>1</code>

Outline

- ✓ Some Tips
- ✓ Conditional
- ✓ Iteration
- ✓ Useful Skills
- ✓ Some Practices



Conditional

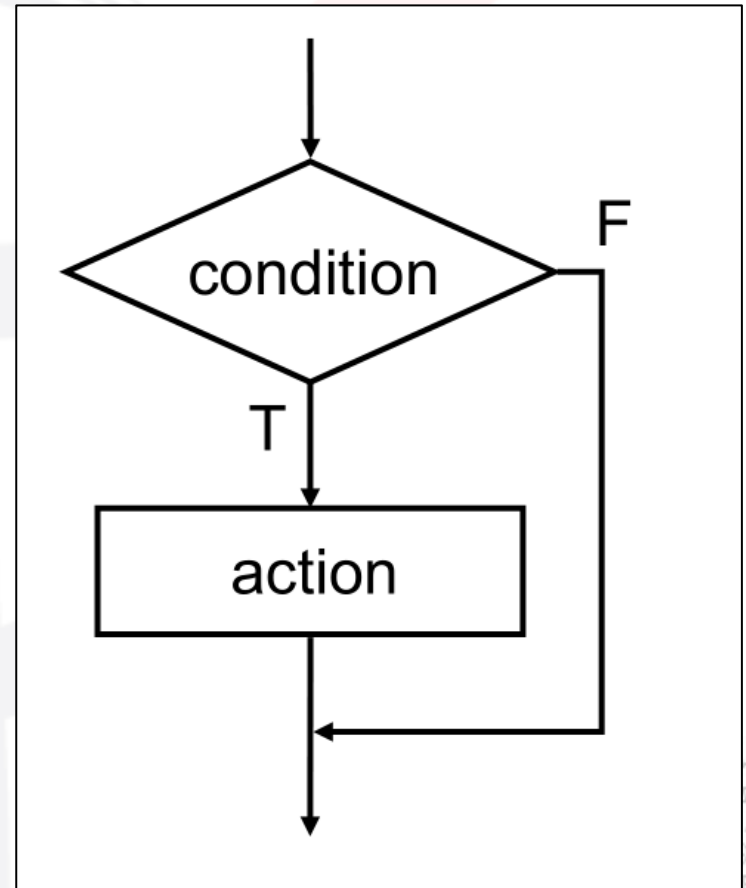
- ✓ IF
- ✓ IF-ELSE
- ✓ SWITCH



If

```
if (condition)
    action;
```

- **Condition** is an **expression**, which evaluates to **TRUE** (*non-zero*) or **FALSE** (*zero*).
- **Action** is a **statement**, which may be **simple** or **compound** (*a block*).



Example if Statements

```
if (x <= 10)
```

```
    y = x * x + 5;
```

simple statement

```
if (x <= 10) {
```

```
    y = x * x + 5;
```

```
    z = (2 * y) / 3;
```

```
}
```

如果 $x \leq 10$ ，兩行都會被執行；
否則，兩行都會被跳過。

compound statement

```
if (x <= 10)
```

```
    y = x * x + 5;
```

```
    z = (2 * y) / 3;
```

如果 $x \leq 10$ ，這行會被執行；
否則，這行會被跳過。

simple statement

simple statement

這行一定會被執行！



More if Examples

```
if (0 <= age && age <= 11)
    kids += 1;
```

```
if (month == 4 || month == 6 || month == 9 ||
    month == 11)
    cout << "The month has 30 days.\n";
```

```
if (x = 2) ← Always true!!
    y = 5; ← 這行一定會被執行!!
```

這是相當常見的錯誤 (= 寫成 ==),
但不會編譯錯誤, 因為語法正確 (syntactically correct) !
(語意錯誤)

A common pitfall



if's can be Nested (巢狀的)

```
if (x == 3)
    if (y != 6) {
        z = z + 1;
        w = w + 2;
    }
```

is the same as...

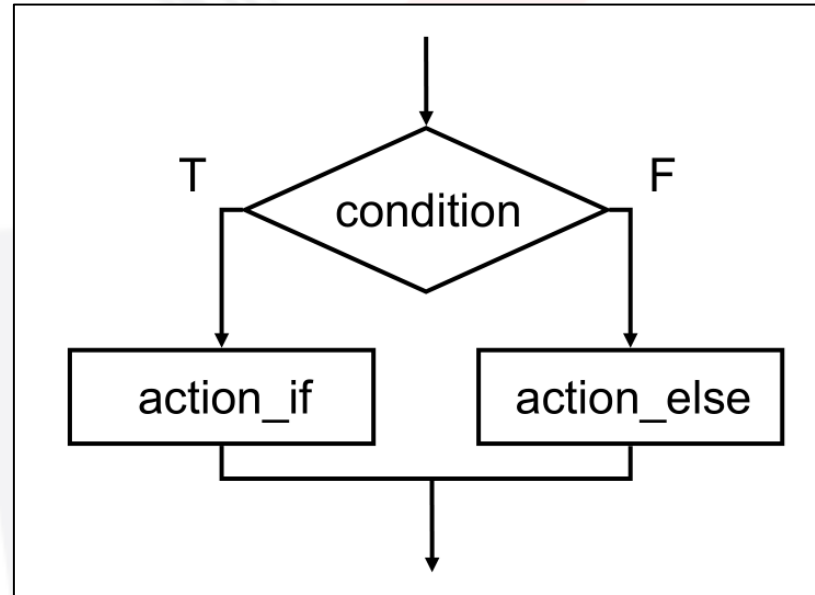
```
if (x == 3) {
    if (y != 6) {
        z = z + 1;
        w = w + 2;
    }
}
```

```
if ((x == 3) && (y != 6)) {
    z = z + 1;
    w = w + 2;
}
```



If-else

```
if (condition)
    action_if;
else
    action_else;
```



- *else* 允許在兩個互斥的 *action* 中選擇，而不需要重新測試(*another if*)。
- ***condition*: an *expression***
- ***action_if, action_else*: a *statements***



Matching Else with If

```
if (x != 10)
  if (y > 3)
    z = z / 2;
  else
    z = z * 2;
```

is the same as...

```
if (x != 10) {
  if (y > 3)
    z = z / 2;
  else
    z = z * 2;
}
```

else 只與「**最接近**」的未被配對的 *if* 做配對！

is NOT the same as...

```
if (x != 10) {
  if (y > 3)
    z = z / 2;
}
else
  z = z * 2;
```



Chaining If's and Else's (If – Elseif – Else)

```
if (month == 4 || month == 6 || month == 9 || month == 11)
    cout << "Month has 30 days.\n";
else if (month == 1 || month == 3 || month == 5 ||
        month == 7 || month == 8 || month == 10 ||
        month == 12)
    cout << "Month has 31 days.\n";
else if (month == 2)
    cout << "Month has 28 or 29 days.\n";
else
    cout << "Don't know that month.\n";
```



If – Elseif – Else vs All ifs (1/2)

- Compared to the previous page, what happened is month == 4?

```
if (month == 4 || month == 6 || month == 9 || month == 11)
    cout << "Month has 30 days.\n";
if (month == 1 || month == 3 || month == 5 ||
    month == 7 || month == 8 || month == 10 ||
    month == 12)
    cout << "Month has 31 days.\n";
if (month == 2)
    cout << "Month has 28 or 29 days.\n";
else
    cout << "Don't know that month.\n";
```

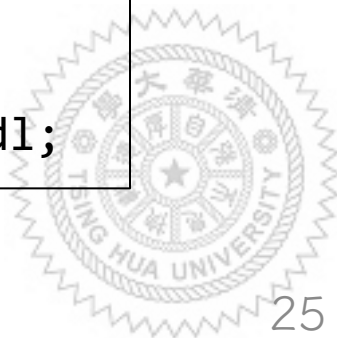


If – Elseif – Else vs All ifs (2/2)

- Now a is 3, b is 6, and c is 9.
- Output?

```
if (a == 3)
    cout << a << endl;
else if (b == 6)
    cout << b << endl;
else if (c == 9)
    cout << c << endl;
else
    cout << "?" << endl;
```

```
if (a == 3)
    cout << a << endl;
if (b == 6)
    cout << b << endl;
if (c == 9)
    cout << c << endl;
else
    cout << "?" << endl;
```



If-Else vs Conditional operator ? :

```
if (x != 0)  
    y = 0;  
else  
    y = 1;
```

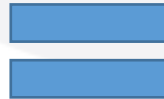


```
if (x)  
    y = 0;  
else  
    y = 1;
```



y = !x;

```
y = (x != 0) ? 0 : 1;
```



```
y = x ? 0 : 1;
```



if (false) ?

- 根據定義
- `if (false) {`
...
`}`
- 大括號內都會跳過
- 有意義嗎？



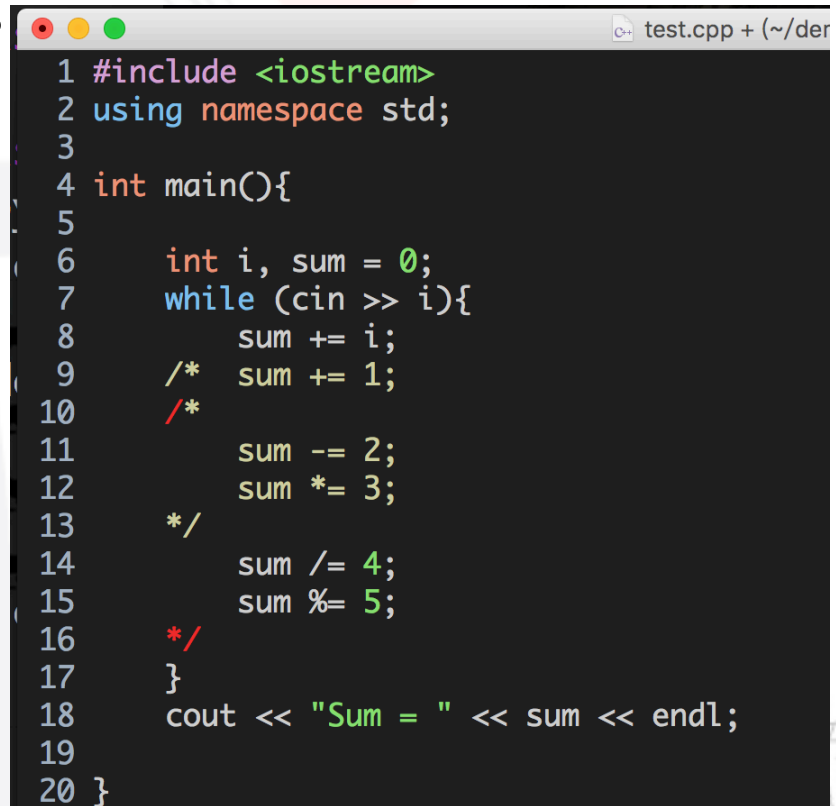
Why if (false) ? (1/3)

- 意義當然是有的
- 假設你不想要第 9~15 行，但不想刪掉以免以後會再用到，因此用註解的方式。
- e.g. 想要註解第 9~15 行，但第 10~13 行已經註解掉了。
- 當然可以將 `/*`, `*/` 刪掉放到第 9 和第 15 行

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5
6     int i, sum = 0;
7     while (cin >> i){
8         sum += i;
9         sum += 1;
10        /*
11            sum -= 2;
12            sum *= 3;
13        */
14            sum /= 4;
15            sum %= 5;
16        }
17        cout << "Sum = " << sum << endl;
18
19 }
```

Why if (false) ? (2/3)

- e.g. 想要註解第 9~15 行，
但第 10~13 行已經註解掉了。
- 直接在第 9 和第 15 行註解
可以嗎？
- 不可以！`/* */` 不能巢狀註
解



```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5
6     int i, sum = 0;
7     while (cin >> i){
8         sum += i;
9         /* sum += 1;
10        /*
11         sum -= 2;
12         sum *= 3;
13        */
14         sum /= 4;
15         sum %= 5;
16        */
17     }
18     cout << "Sum = " << sum << endl;
19
20 }
```

Why if (false) ? (3/3)

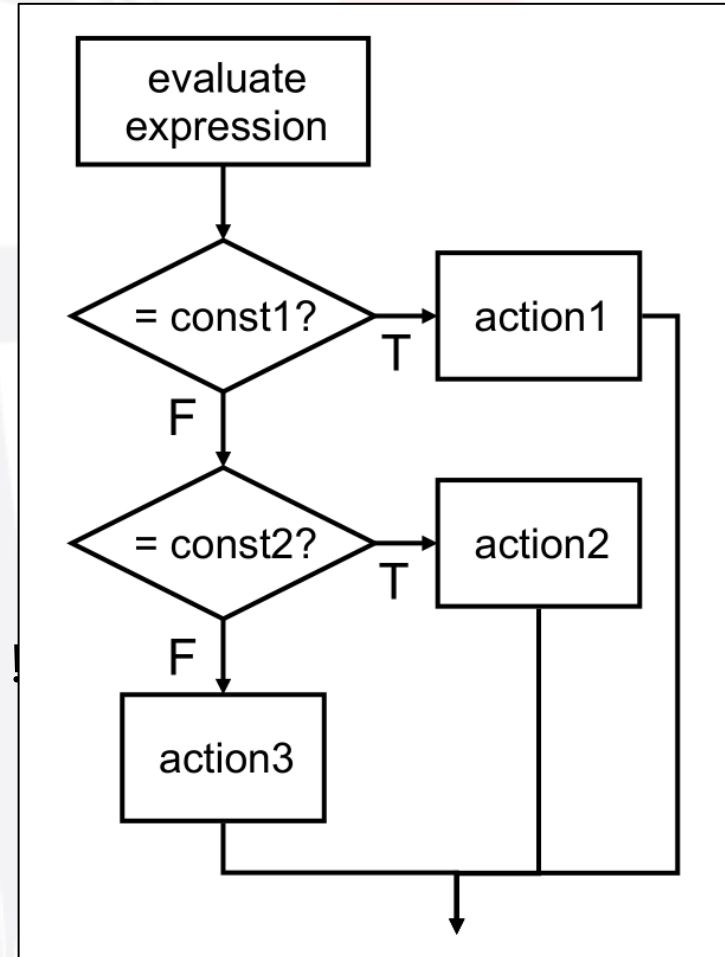
- e.g. 想要註解第 9~15 行，
但第 10~13 行已經註解掉了。
- Alternative way: 用
if (0) { } 包住
- 則裡面的都不會被執行。
- 等等教迴圈後 while (0)
{ } 也可註解。

```
test.cpp + (~/  
1 #include <iostream>  
2 using namespace std;  
3  
4 int main(){  
5  
6     int i, sum = 0;  
7     while (cin >> i){  
8         sum += i;  
9         if (0){  
10             sum += 1;  
11             /*  
12                 sum -= 2;  
13                 sum *= 3;  
14             */  
15             sum /= 4;  
16             sum %= 5;  
17         }  
18     }  
19     cout << "Sum = " << sum << endl;  
20  
21 }
```

Switch

```
switch (expression) {  
    case const1:  
        action1;  
        break;  
    case const2:  
        action2;  
        break;  
    default:  
        action3;  
}
```

- 長的 if-elseif...-else 的替代方案！
- 注意！ 如果沒有 break，那麼會繼續執行(falls through)到遇到 break 為止！



Switch Example

```
/* same as month example for if-else */
switch (month) {
    case 4:
    case 6:
    case 9:
    case 11: /* month == 4 || month == 6 || month == 9 || month == 11 */
        cout << "Month has 30 days.\n";
        break;
    case 1: case 3: case 5: case 7:
    case 8: case 10: case 12:
        cout << "Month has 31 days.\n";
        break;
    case 2:
        cout << "Month has 28 or 29 days.\n";
        break;
    default:
        cout << "Don't know that month.\n";
}
```



More About Switch (1/2)

- Switch expression 一定要是**整數變數 (或字元變數)**，不能是浮點數變數或字串變數。

switch (x){ ... } // float x; is illegal

- Case expressions 一定要是**整數常數 (或字元)**，不能是浮點數或字串。

case i: /* 若 i 是變數就不合法 */
/* 例如：若宣告 int i = 0; 不合法！
但若宣告 const int i = 0;
則合法！ */



More About Switch (2/2)

➤ 若沒寫 `break;` 下一個 `case` 也會被執行！

```
switch (a) {  
    case 1:  
        cout << "A";  
    case 2:  
        cout << "B";  
    default:  
        cout << "C";  
}
```

若 `a == 1`, prints "ABC".
若 `a == 2`, prints "BC".
否則, prints "C".



Outline

- ✓ Some Tips
- ✓ Conditional
- ✓ **Iteration**
- ✓ Useful Skills
- ✓ Some Practices



Iteration

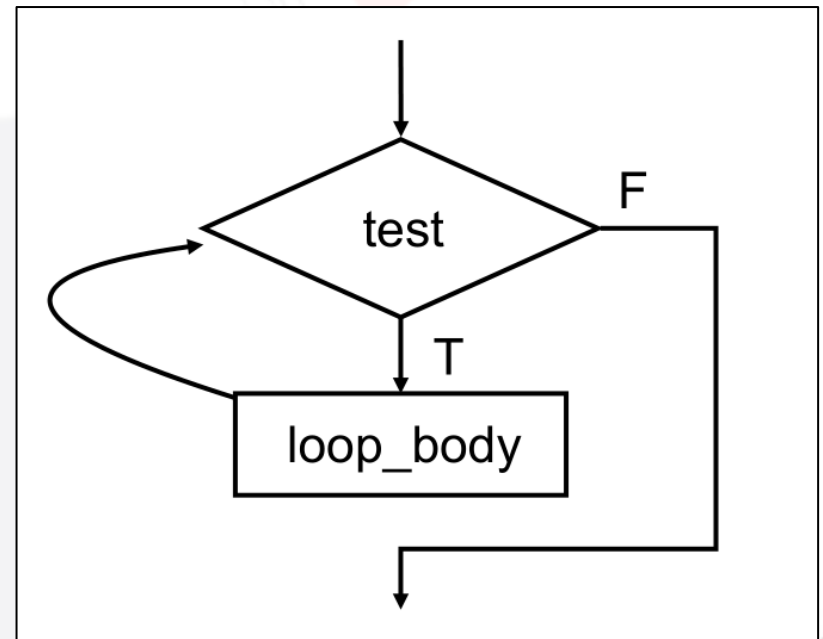
- ✓ WHILE
- ✓ FOR
- ✓ DO-WHILE



While

```
while (test)
    loop_body;
```

- *test*: C++ *expression*
- *loop_body*: C++ *statement*
- *test* 是 *TRUE* (*non-zero*)，就執行 *loop_body*，執行完再回到 *test*。
- *test* 是 *FALSE* (*zero*)，就離開迴圈。
- *Note*: *test* is evaluated *before* 執行 *loop_body*.
- while vs. do-while



Example While Loops

```
int i = 0;
while (i < 5){
    cout << i << ' ';
    i++;
}
cout << endl; // why this line?
```

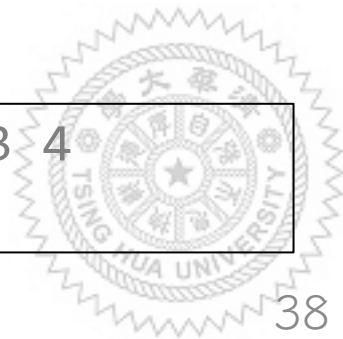
0 1 2 3 4

```
int i = 0;
while (i++ < 5)
    cout << i << ' ';
cout << endl;
```

1 2 3 4 5

```
int i = 0;
while (i < 5)
    cout << i++ << ' ';
cout << endl;
```

0 1 2 3 4



Handle Different Input Types (1/3)

- 我們要寫一段給很多數字算總和的程式碼 (有很多種 input 格式)

1. 有給 input 個數
(input 第一個數字
代表 input 個數)

```
Slighten@SlightenCheng ttys000-bash 04:32 ~ $ ./a.out
5
1 3 5 7 9
Sum = 25
```

```
int i, num, sum = 0;
cin >> num;
while (num-- > 0){
    cin >> i;
    sum += i;
}
cout << "Sum = "
    << sum << endl;
```

```
int i, j = 0, num, sum = 0;
cin >> num;
while (j++ < num){
    cin >> i;
    sum += i;
}
cout << "Sum = "
    << sum << endl;
```

Handle Different Input Types (2/3)

2. 沒有給 input 個數 (按 Ctrl-D 代表檔尾)

```
int i, sum = 0;
while (cin >> i)
    sum += i;
cout << "Sum = " << sum << endl;
```

```
Slighten@SlightenCheng ttys000-bash 04:38 ~ $ ./a.out
1 3 5 7 9
Sum = 25
```

Handle Different Input Types (3/3)

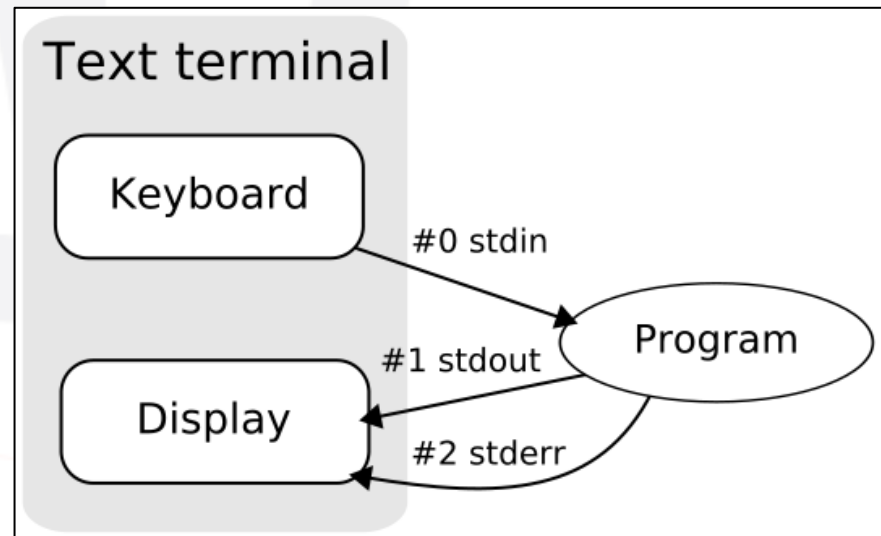
- 問題：如果 input 有 1000 組數字？要一個一個手動輸入嗎？(例如 Uva 的題目)
- No! 將 input 存進檔案讀檔就好！
- 讀檔有兩種方式：
 1. main function 吃參數 (檔案名稱) + C/C++ 檔案讀寫 (Chapter 4, 6) → 要改程式碼
 2. 用 Unix 資料重導向 (I/O Redirection) → 程式碼完全不用改

- 更多資訊參照鳥哥的 Linux 私房菜: http://linux.vbird.org/linux_basic/0320bash.php



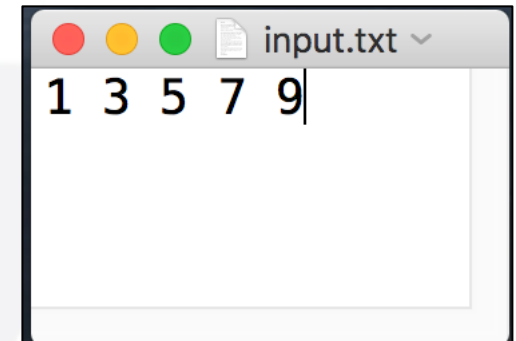
3 Types of Standard I/O

- `stdin` (`cin >>`)
 - 預設從鍵盤讀取
- `stdout` (`cout <<`)
 - 預設輸出至螢幕
- `stderr` (`cerr <<`)
 - 預設輸出至螢幕
- 我們可以利用重導向技巧更改預設輸入輸出的位置



How to Use Input Redirection

- 先將 input 存進 input.txt，放在執行檔旁邊
- `% ./a.out < input.txt` # % is a prompt
- < 運算子的意義是將右邊的檔案的內容取代標準輸入(鍵盤輸入)

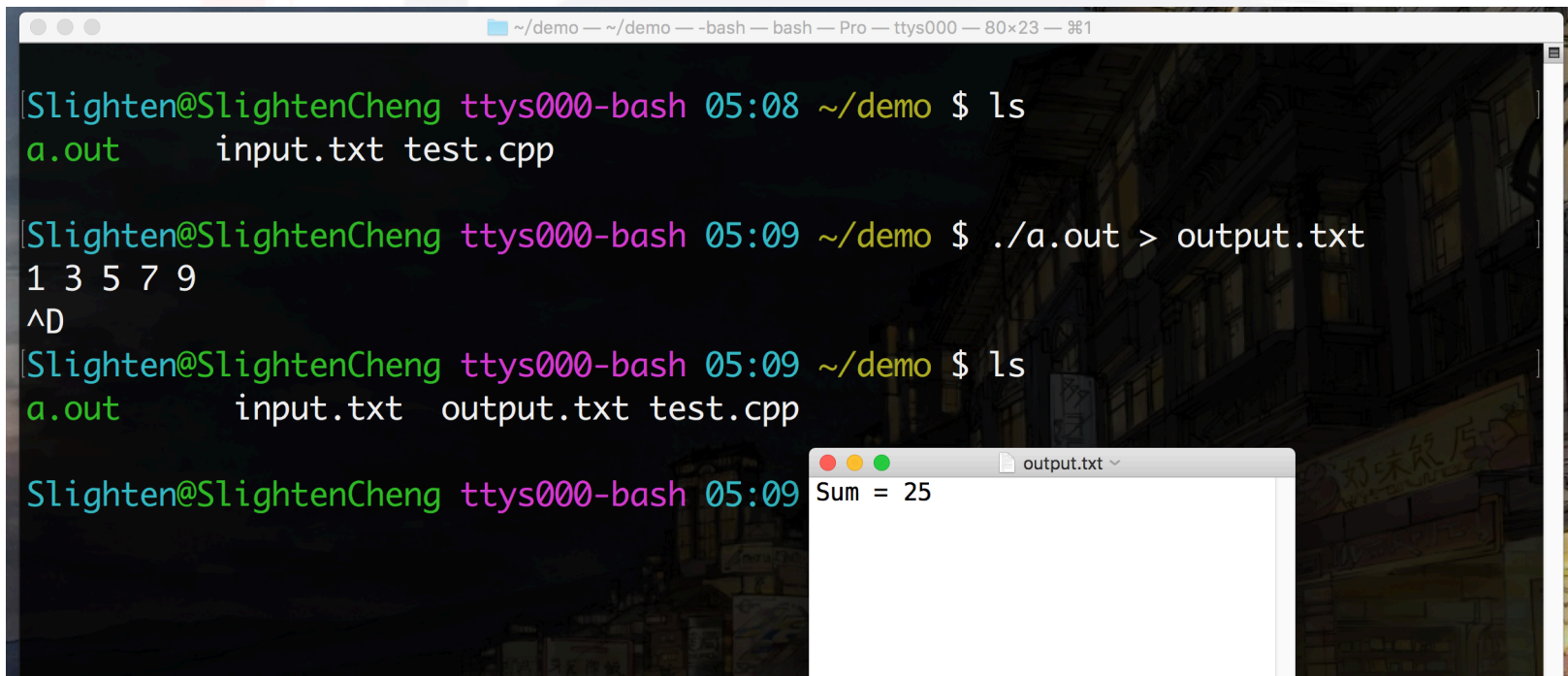


```
Slighten@SlightenCheng ttys000-bash 04:56 ~/demo $ ls  
a.out      input.txt test.cpp
```

```
Slighten@SlightenCheng ttys000-bash 04:56 ~/demo $ ./a.out < input.txt  
Sum = 25
```


How to Use Output Redirection

- `% ./a.out > output.txt` # % is a prompt
- > 運算子的意義是將右邊的檔案的內容取代標準輸出(螢幕輸出)



The image shows a terminal window with a dark background and a light-colored text cursor. The terminal title bar reads: `~/demo — ~/demo — -bash — bash — Pro — ttys000 — 80x23 — 881`. The terminal content shows the following sequence of commands and output:

```
Slighten@SlightenCheng ttys000-bash 05:08 ~/demo $ ls
a.out      input.txt  test.cpp

Slighten@SlightenCheng ttys000-bash 05:09 ~/demo $ ./a.out > output.txt
1 3 5 7 9
^D

Slighten@SlightenCheng ttys000-bash 05:09 ~/demo $ ls
a.out      input.txt  output.txt  test.cpp

Slighten@SlightenCheng ttys000-bash 05:09
```

In the bottom right corner, there is a small window titled `output.txt` showing the output of the program: `Sum = 25`.

How to Use I/O Redirection Simultaneously

- % `./a.out < input.txt > output.txt`
- 不一定要用純文字編輯器看檔案，在 Linux 上可以用 `cat` 指令將檔案內容印到螢幕上 (`cat -n filename` 可顯示行號)

```
Slighten@SlightenCheng ttys000-bash 05:15 ~/demo $ ls
a.out      input.txt test.cpp

Slighten@SlightenCheng ttys000-bash 05:15 ~/demo $ ./a.out < input.txt > output.txt

Slighten@SlightenCheng ttys000-bash 05:15 ~/demo $ ls
a.out      input.txt  output.txt test.cpp

Slighten@SlightenCheng ttys000-bash 05:15 ~/demo $ cat input.txt
1 3 5 7 9
Slighten@SlightenCheng ttys000-bash 05:17 ~/demo $ cat output.txt
Sum = 25
```

用 diff 指令比較兩個檔案的差異 (1/5)

- 想要比較你的輸出是否與「標準答案」一致？
- 將標準答案存在 answer.txt
- % diff output.txt answer.txt
- 如果**完全沒有螢幕輸出**那就是兩個檔案「**一模一樣**」
- 否則螢幕上會顯示兩個檔案差異之處！



用 diff 指令比較兩個檔案的差異 (2/5)

- 第2~4行不同 (多一個空白、換行、縮排也算不同)

```
Slighten@SlightenCheng ttys000-bash 05:30 ~/demo2 $ ls
answer.txt output.txt

Slighten@SlightenCheng ttys000-bash 05:30 ~/demo2 $ cat answer.txt
1
1 2
1 2 3
1 2 3 4

Slighten@SlightenCheng ttys000-bash 05:30 ~/demo2 $ cat output.txt
1
12
1 2 2
1 2 3 4

Slighten@SlightenCheng ttys000-bash 05:30 ~/demo2 $ diff output.txt answer.txt
2,4c2,4
< 12
< 1 2 2
< 1 2 3 4
---
> 1 2
> 1 2 3
> 1 2 3 4
```



用 diff 指令比較兩個檔案的差異 (3/5)

- 想要有顏色的 diff 看看有沒有 colordiff 指令。
- 沒有就 % brew install colordiff

```
Slighten@SlightenCheng ttys000-bash 05:30 ~/demo2 $ diff output.txt answer.txt
2,4c2,4
< 12
< 1 2 2
< 1 2 3 4
---
> 1 2
> 1 2 3
> 1 2 3 4

Slighten@SlightenCheng ttys000-bash 05:31 ~/demo2 $ colordiff output.txt answer.txt
2,4c2,4
< 12
< 1 2 2
< 1 2 3 4
---
> 1 2
> 1 2 3
> 1 2 3 4
```



用 diff 指令比較兩個檔案的差異 (4/5)

- 當兩個檔案相同時就沒有螢幕輸出。

```
Slighten@SlightenCheng ttys000-bash 05:36 ~/demo2 $ ls
answer.txt output.txt

Slighten@SlightenCheng ttys000-bash 05:36 ~/demo2 $ cp answer.txt output2.txt # 將answer.txt複製並改名成output2.txt貼上

Slighten@SlightenCheng ttys000-bash 05:38 ~/demo2 $ ls
answer.txt output.txt output2.txt

Slighten@SlightenCheng ttys000-bash 05:38 ~/demo2 $ cat answer.txt
1
1 2
1 2 3
1 2 3 4

Slighten@SlightenCheng ttys000-bash 05:38 ~/demo2 $ cat output2.txt
1
1 2
1 2 3
1 2 3 4

Slighten@SlightenCheng ttys000-bash 05:38 ~/demo2 $ diff answer.txt output2.txt

Slighten@SlightenCheng ttys000-bash 05:38 ~/demo2 $
```

用 diff 指令比較兩個檔案的差異 (5/5)

- 回憶作業 hw1_3.cpp 要輸出 hw1_1.cpp 的程式碼。
- 請執行
- `% g++ hw1_3.cpp`
- `% ./a.out > output.txt`
- `% diff output.txt hw1_1.cpp`
- 看輸出是否與 hw1_1.cpp 一樣。
- 想一想如果有差是差在哪呢？（例如縮排應該 `cout << '\t'` 但你卻用四個空白 `cout << " "`）



無窮迴圈 (Infinite Loops) (1/2)

- 以下迴圈永遠不會終止:

```
int x = 0;  
while (x < 10)  
    cout << x << endl;
```



A common pitfall

- *loop_body* 沒有改變 *x* 值，所以 *test* 永遠不會失敗 (always **true**).
- 這是相當常見但不好發現的錯誤
- 並不會編譯錯誤，因為語法正確 (syntactically correct) ! (語意錯誤)
- 怎麼辦？利用 Ctrl-C (hold Ctrl key and type C) to kill it.

無窮迴圈 (Infinite Loops) (2/2)

但我們有時候會想要無窮迴圈，比如說計算機或有很多個回合的遊戲。

```
while (1) {  
    /* Do what you want to do */  
}
```

一樣利用 Ctrl-C (hold Ctrl key and type C) to kill it.
(Or use `break` in *loop_body* to exit the while loop.)



Use of While Loops in input/output

```
char echo = 'A';  
while (echo != '\n') {  
    cin >> echo;  
    cout << echo;  
}
```

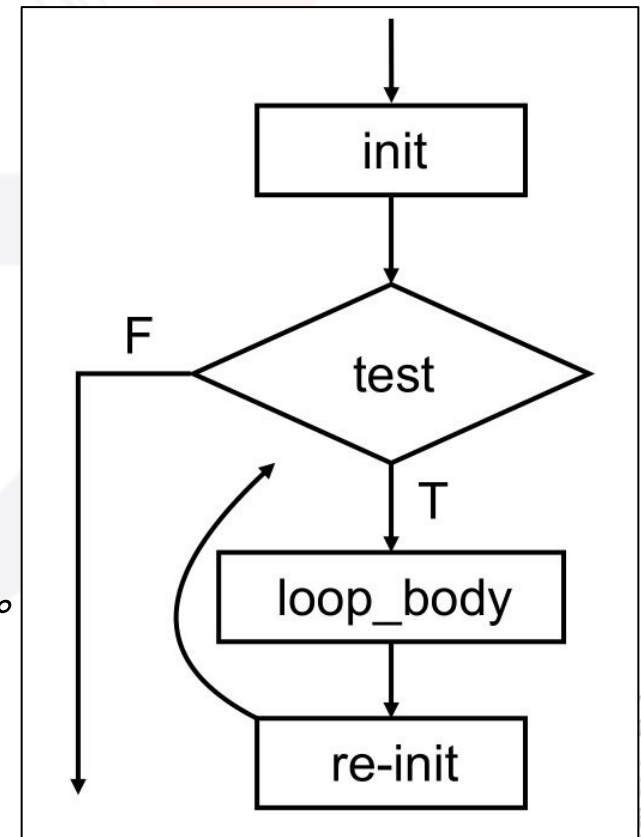
- 會發生什麼事？
- 答案是不會離開 while loop，因為如果 cin 後接的參數是 `int` or `char`，會跳過所有的 `tab` (`'\t'`), `whitespace` (`' '`), and `newline` (`'\n'`)。



For

```
for (init; test; re-init)
    loop_body
```

- *init, test, re-init*: C++ expressions
- *loop_body*: C++ statement
- *init* 初始化 **before** *test*
- *re-init* 再初始化 **after** *loop_body*
- *test* 是 **TRUE** (*non-zero*), 就執行 *loop_body*, 執行完再初始化後, 再回到 *test*。
- *test* 是 **FALSE** (*zero*), 就離開迴圈。
- Note: *test* is evaluated **before** 執行 *loop_body*.
- while vs. do-while vs. for



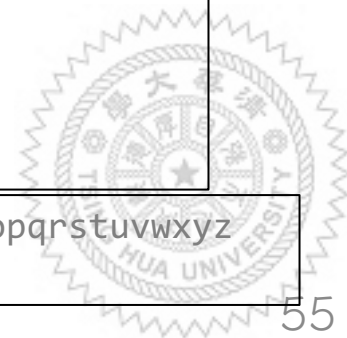
Example For Loops

```
/* -- what is the output of this loop? -- */  
int i;  
for (i = 0; i <= 10; i++)  
    cout << i << ' ';  
cout << endl; // why this line?
```

0 1 2 3 4 5 6 7 8 9 10

```
/* -- what does this one output? Recall the ASCII table. -- */  
char letter = 'a';  
int c;  
for (c = 0; c < 26; c++)  
    cout << (char) (letter + c); // Recall type-casting  
cout << endl; // why this line?
```

abcdefghijklmnopqrstuvwxyz



The Variable After For Loop

```
int i;  
for (i = 0; i <= 10; i++)  
    cout << i << ' '  
cout << "The value of i after the for loop is: "  
    << i << endl;    // what is the answer?
```

- 答案是 **11**，因為 $i = 10$ 後滿足 $test (i \leq 10)$ ，進入 *Loop_body*，印出 **10** 後，做 *re-init* ($i++$)， i 變成 **11**，不滿足 $test (i \leq 10)$ ，離開 for loop

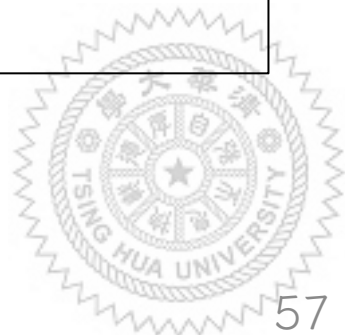


宣告變數的時機與差異？

```
int i;  
for (i = 0; i <= 10; i++)  
    cout << i << ' ';  
cout << "The value of i after the for loop is: "  
    << i << endl;    // what is the answer?
```

```
for (int i = 0; i <= 10; i++)  
    cout << i << ' ';  
cout << "The value of i after the for loop is: "  
    << i << endl;    // what is the answer?
```

這裡會發生**編譯錯誤**，因為變數 `i` 的生存時間只在 for loop 裡！！
一離開 for loop，變數 `i` 就消失了！



Nested Loops (巢狀迴圈)

- Loop body 當然可以是另一個迴圈！

```
for (int i = 1; i < 10; i++) {  
    for (int j = 1; j < 10; j++) {  
        cout << i*j << '\t';  
    }  
    cout << endl; // Why this line here?  
}
```

可省略(Why?)，寫出來只是為了好閱讀！

is the same as...

```
for (int i = 1; i < 10; i++) {  
    for (int j = 1; j < 10; j++)  
        cout << i*j << '\t';  
    cout << endl; // Why this line here?  
}
```

outer loop

inner loop

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

Example Nested Loops -- 九九乘法表

```
for (int i = 1; i < 10; i++) {  
    for (int j = 1; j < 10; j++)  
        cout << j << " x " << i << " = "  
            << setw(2) << i*j << '\t';  
    cout << endl;  
}
```

setw(2)? 這是什麼?

注意 i, j 的位置, 想一想 i 跟 j 交換會怎樣?

1 x 1 = 1	2 x 1 = 2	3 x 1 = 3	4 x 1 = 4	5 x 1 = 5	6 x 1 = 6	7 x 1 = 7	8 x 1 = 8	9 x 1 = 9
1 x 2 = 2	2 x 2 = 4	3 x 2 = 6	4 x 2 = 8	5 x 2 = 10	6 x 2 = 12	7 x 2 = 14	8 x 2 = 16	9 x 2 = 18
1 x 3 = 3	2 x 3 = 6	3 x 3 = 9	4 x 3 = 12	5 x 3 = 15	6 x 3 = 18	7 x 3 = 21	8 x 3 = 24	9 x 3 = 27
1 x 4 = 4	2 x 4 = 8	3 x 4 = 12	4 x 4 = 16	5 x 4 = 20	6 x 4 = 24	7 x 4 = 28	8 x 4 = 32	9 x 4 = 36
1 x 5 = 5	2 x 5 = 10	3 x 5 = 15	4 x 5 = 20	5 x 5 = 25	6 x 5 = 30	7 x 5 = 35	8 x 5 = 40	9 x 5 = 45
1 x 6 = 6	2 x 6 = 12	3 x 6 = 18	4 x 6 = 24	5 x 6 = 30	6 x 6 = 36	7 x 6 = 42	8 x 6 = 48	9 x 6 = 54
1 x 7 = 7	2 x 7 = 14	3 x 7 = 21	4 x 7 = 28	5 x 7 = 35	6 x 7 = 42	7 x 7 = 49	8 x 7 = 56	9 x 7 = 63
1 x 8 = 8	2 x 8 = 16	3 x 8 = 24	4 x 8 = 32	5 x 8 = 40	6 x 8 = 48	7 x 8 = 56	8 x 8 = 64	9 x 8 = 72
1 x 9 = 9	2 x 9 = 18	3 x 9 = 27	4 x 9 = 36	5 x 9 = 45	6 x 9 = 54	7 x 9 = 63	8 x 9 = 72	9 x 9 = 81

I/O manipulation

```
#include <iomanip>
std::setw(int n);
```

- 功能：為下一個 cout 的元素留下寬度為 n 個字元的空間，然後下一個元素靠右邊開始放。

setw() 參考：

<http://www.cplusplus.com/reference/iomanip/setw/>

iomanip.h 參考：

<http://www.cplusplus.com/reference/iomanip/>



Example of using setw()

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  int main(int argc, char **argv){
6      cout << 1 << endl;
7      cout << setw(5) << 1 << endl;
8      cout << 11 << endl;
9      cout << setw(5) << 11 << endl;
10     cout << 111 << endl;
11     cout << setw(5) << 111 << endl;
12     return 0;
13 }
```

The output of the program is a triangular pattern of numbers on a black background. The first row contains a single '1'. The second row contains a '1' followed by four spaces and another '1'. The third row contains '11'. The fourth row contains '11' followed by four spaces and another '11'. The fifth row contains '111'. The sixth row contains '111' followed by four spaces and another '111'. The seventh row consists of seven dashes ('-----').



Another Nested Loop

- inner loop 的 test 也可以與 outer loop 的變數有關！

```
for (int i = 0; i < 10; i++) {  
    for (int j = 0; j <= i; j++)  
        cout << '(' << i << ',' << j << ')';  
    cout << endl;  
}
```

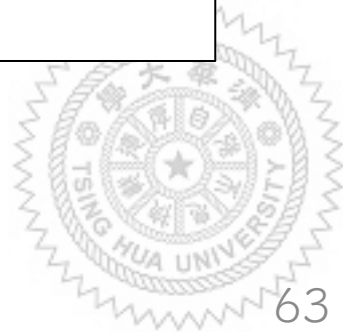
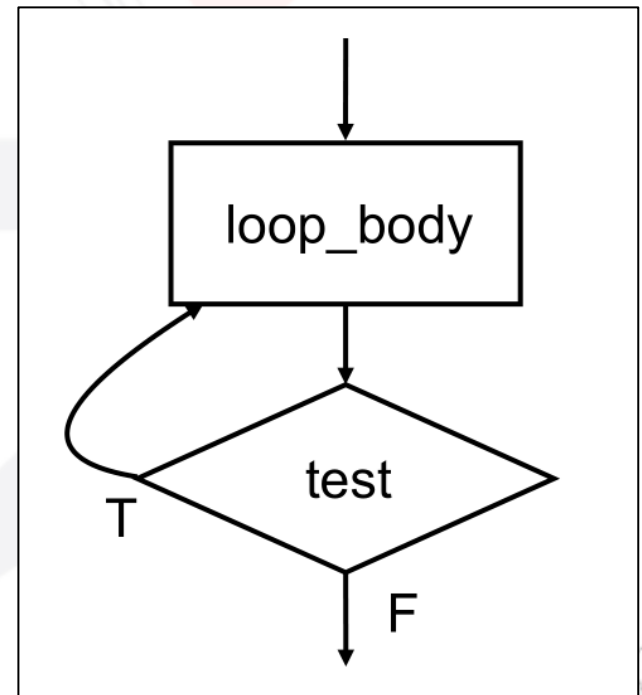
```
(0,0)  
(1,0)(1,1)  
(2,0)(2,1)(2,2)  
(3,0)(3,1)(3,2)(3,3)  
(4,0)(4,1)(4,2)(4,3)(4,4)  
(5,0)(5,1)(5,2)(5,3)(5,4)(5,5)  
(6,0)(6,1)(6,2)(6,3)(6,4)(6,5)(6,6)  
(7,0)(7,1)(7,2)(7,3)(7,4)(7,5)(7,6)(7,7)  
(8,0)(8,1)(8,2)(8,3)(8,4)(8,5)(8,6)(8,7)(8,8)  
(9,0)(9,1)(9,2)(9,3)(9,4)(9,5)(9,6)(9,7)(9,8)(9,9)
```



Do-While

```
do  
    loop_body;  
while (test);
```

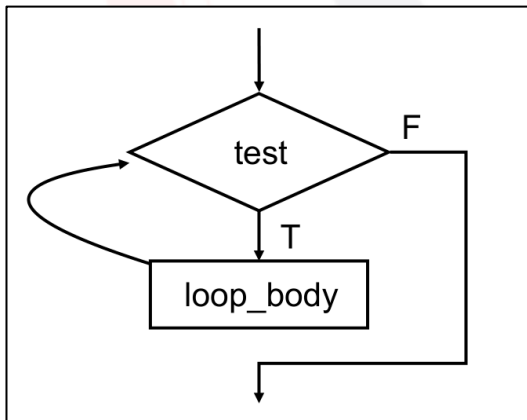
- *test*: C++ *expression*
- *loop_body*: C++ *statement*
- *test* 是 *TRUE* (*non-zero*)，就回去執行 *loop_body*。
- *test* 是 *FALSE* (*zero*)，就離開迴圈。
- Note: *test* is evaluated **after** 執行 *loop_body*.
- do-while vs while



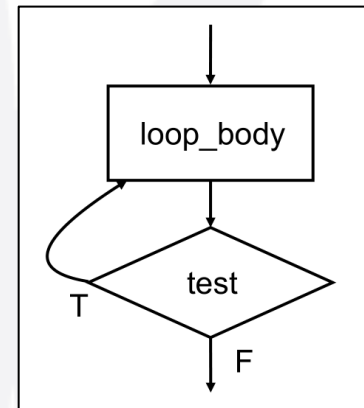
While vs Do-While

- while: 先判斷再做。
- do-while: 先做(一次)再判斷(要不要重做)。

```
while (test)  
    loop_body;
```



```
do  
    loop_body;  
while (test);
```



While vs Do-While

```
int i = 10;
while (i < 10) {
    cout << "enter the loop.\n";
    i++;
}
cout << "After while, i = "
    << i << endl;
```

```
After while, i = 10
```

```
int i = 10;
do {
    cout << "enter the loop.\n";
    i++;
} while (i < 10);
cout << "After do-while, i = "
    << i << endl;
```

```
enter the loop.
After do-while, i = 11
```



For vs While

通常：

- For loop 比較傾向於 counter-based.
 - 明顯的 counter 變數
 - 容易看見每一圈 counter 是如何被修改的
- While loop 比較傾向於 sentinel-based.
 - test 檢查 sentinel value
- 兩種可互換



Break and Continue (1/2)

break;

- 跳出「所在這層的」switch 或迴圈
- 只能用在
 1. switch -- 做完一個 case，或在做 case 中途跳出
 2. 迴圈 -- 中途跳出迴圈

continue;

- 只能用在迴圈
- 中途直接跳回迴圈開頭，重新進行 test
- 在 for loop，會先執行 re-initializer，再跳回迴圈開頭，重新進行 test

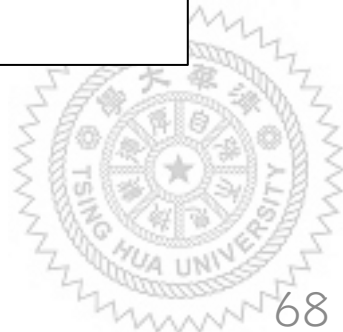
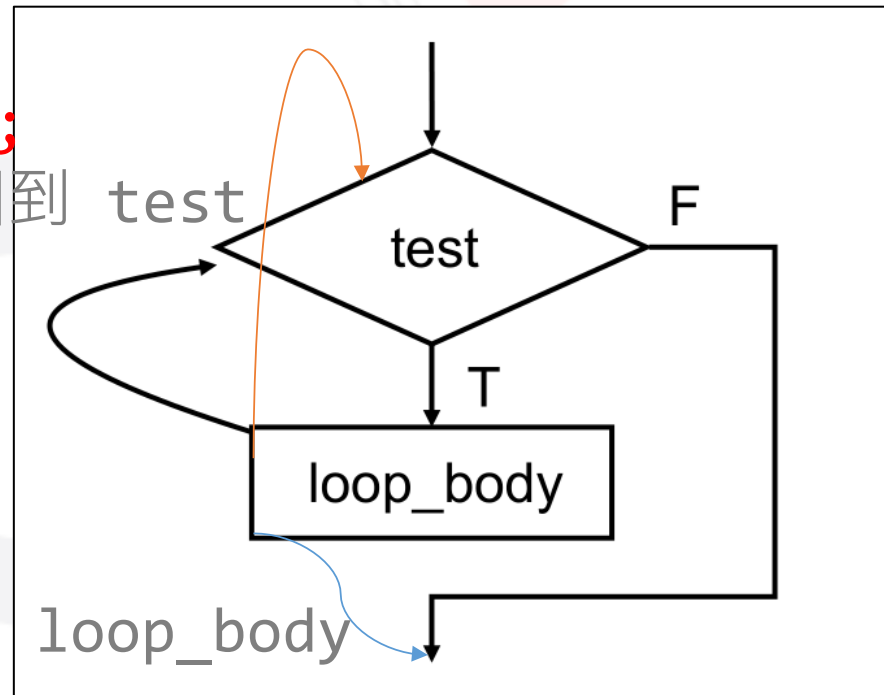


Break and Continue (2/2)

- 以 while 當例子

continue;
// 中途回到 test

break;
// 中途跳出 loop_body



Example

以下迴圈的結果是？

```
for (int i = 0; i <= 20; i++) {  
    if (i % 2 == 0) continue;  
    cout << i << ' ';  
}
```

1 3 5 7 9 11 13 15 17 19

➤如何改寫得更簡單？

```
for (int i = 1; i <= 20; i += 2)  
    cout << i << ' ';
```

➤把 continue; 換成 break; 的結果會是？



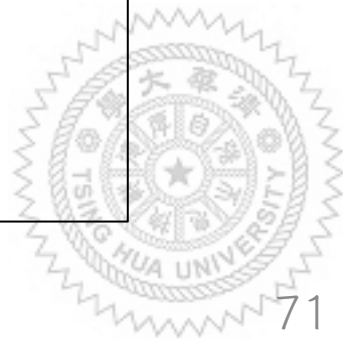
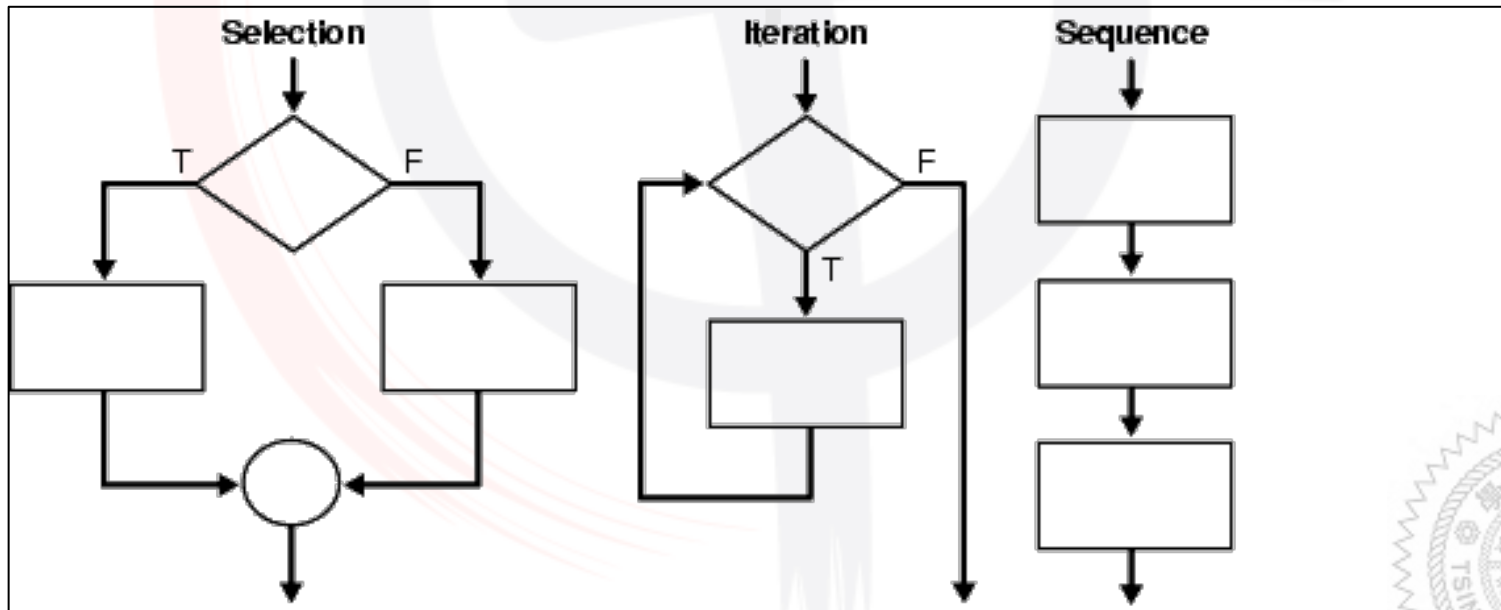
總結 I

- **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



總結 II

- Sequential -- C++ statements
- Conditional -- if-else, switch
- Iterative -- while, for, do-while



Outline

- ✓ Some Tips
- ✓ Conditional
- ✓ Iteration
- ✓ Useful Skills
- ✓ Some Practices



Useful Skills



亂數函式 rand()

- `#include <stdlib>`
- `int rand(void);`
- 功能：產生一個介於 $0 \sim \text{RAND_MAX}$ 的整數亂數。
- 缺點：一但使用了一個亂數種子(seed)就不會再換（第一次是亂數，之後每次都跟第一次一樣）。
- 解法：需要每次給一個不同的亂數種子(ex：現在時間)，
- 利用 `srand(time(NULL));`



亂數種子決定函式 srand()與時間函數 time()

- `#include <cstdlib>`
- `void srand(unsigned int seed);`
- 功能：訂定亂數種子(random seed)。
- `#include <ctime>`
- `time_t time(time_t* timer);`
- 功能：回傳時間戳記(timestamp)。



產生自訂範圍的整數亂數

- `rand()`; 的範圍為 $0 \sim \text{RAND_MAX}$ 。
- 如何將範圍縮到整數 $0 \sim 9$?
- 提示：利用取餘數運算子 `%`
- 答：`rand() % 10;`
- 如何將範圍縮到整數 $1 \sim 10$?
- 答：`rand() % 10 + 1;`
- 如何將範圍縮到整數 $a \sim b$?
- 答：`rand() % (b-a+1) + a;`

•



Example -- Without srand()

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  int main(int argc, char **argv){
7      // srand(time(NULL));
8      for (int i = 0; i < 10 ; i++)
9          cout << rand() % 10 << ' ';
10     cout << endl;
11     return 0;
12 }
```

第一次執行

1 7 4 0 9 4 8 8 2 4

第二次執行

1 7 4 0 9 4 8 8 2 4



Example -- With srand()

```
1 #include <iostream>
2 #include <cstdlib>
3 #include <ctime>
4 using namespace std;
5
6 int main(int argc, char **argv){
7     srand(time(NULL));
8     for (int i = 0; i < 10 ; i++)
9         cout << rand() % 10 << ' ';
10    cout << endl;
11    return 0;
12 }
```

第一次執行

8 3 0 3 0 0 2 9 1 9

第二次執行

3 8 9 3 0 9 8 3 4 6



三種常見的變數類型

1. Counter
2. Flag
3. State (Mode)



Counter

- Counter 變數：用來數次數(正數、倒數)。

```
int lower_case_count = 0; // Counter
char ch;
while (cin >> ch){
    if (ch >= 'a' && ch <= 'z')
        lower_case_count++;
}
cout << "lower case count: "
      << lower_case_count << endl;
```

```
12jDjkYQugk4
^Z
lower case count: 6
```

利用輸入檔尾字元 Ctrl-Z (DOS) 或 Ctrl-D (Linux)，按 Enter 來跳出 while 迴圈



Flag

- Flag 變數：用來表示二元狀態，例如有或無，生或死，找到或未找到.....

```
bool isDead = false; // flag
while(...){
    .
    .
    isDead = (hp <= 0);
    .
    .
}
.
.
if (isDead) ...
```



State (Mode)

- State (Mode) 變數：用來表示多元狀態，例如移動方向(上下左右)，遊戲模式.....

```
int dir = 0; // State => 0: up, 1: right, 2: down, 3: left
char c;
cin >> c;
switch (c){
    case 'w': dir = 0; ... ; break;
    case 'a': dir = 3; ... ; break;
    case 's': dir = 2; ... ; break;
    case 'd': dir = 1; ... ; break;
    default: ...;
}
```



Good Coding Practice

- ✓ 每寫一小段就做測試 (cout)
- ✓ 寫註解
- ✓ 一致的 coding style
 - 用 { }，一致的變數、函式命名
- ✓ 模組化
 - 副函式化 (next chapter)
 - 使得能單獨進行測試 (unit-test)
- ✓ 避免全域變數
- ✓ 考慮邊界條件
 - 正 or 負 or 0，整數 or 非整數等……



Outline

- ✓ Some Tips
- ✓ Conditional
- ✓ Iteration
- ✓ Useful Skills
- ✓ Some Practices



Some Practices



作業 Part I

Portrait ASCII Table

- 檔名叫作 ASCII.cpp
- 印出所有 printable set of characters (32 to 127) of the ASCII table.
- You may use nested loops.
- Think how do the outer and the inner counter go.

```
for (int i = ?; i < ?; i++) {  
    for (int j = ?; j < ?; j++)  
        cout << ? << '\t';  
    cout << endl;  
}
```

- You may use type casting.
 - try `int i = 65; cout << (char) i;`



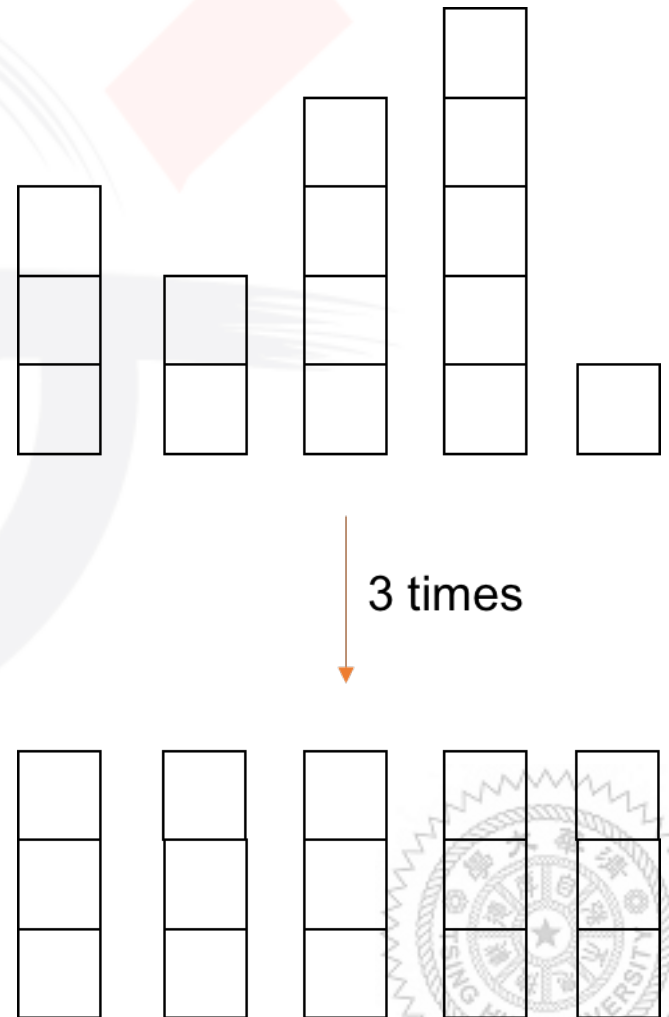
Output

```
32: 64: @ 96: `
33: ! 65: A 97: a
34: " 66: B 98: b
35: # 67: C 99: c
36: $ 68: D 100: d
37: % 69: E 101: e
38: & 70: F 102: f
39: ' 71: G 103: g
40: ( 72: H 104: h
41: ) 73: I 105: i
42: * 74: J 106: j
43: + 75: K 107: k
44: , 76: L 108: l
45: - 77: M 109: m
46: . 78: N 110: n
47: / 79: O 111: o
48: 0 80: P 112: p
49: 1 81: Q 113: q
50: 2 82: R 114: r
51: 3 83: S 115: s
52: 4 84: T 116: t
53: 5 85: U 117: u
54: 6 86: V 118: v
55: 7 87: W 119: w
56: 8 88: X 120: x
57: 9 89: Y 121: y
58: : 90: Z 122: z
59: ; 91: [ 123: {
60: < 92: \ 124: |
61: = 93: ] 125: }
62: > 94: ^ 126: ~
63: ? 95: _ 127: ^
```

作業 Part II

Move Boxes

- 檔名叫作 `movebox.cpp`
- 有 n 堆箱子，有一個搬箱子工人，想計算至少需要搬動幾次，才能讓每堆箱子的數目都一樣。
- Input: 第一個數字是堆數 (n)，第二個數字是箱子總數 (m)，保證 $m \% n == 0$ ，後面 n 個數字代表每堆箱子的個數 (總和 $== m$)
- Output: 最少搬動次數 `#move`
`= ???`



Simple Testcases

```
Slighten@SlightenCheng ttys000-bash 06:28 ~/demo $ g++ movebox.cpp
```

```
Slighten@SlightenCheng ttys000-bash 06:28 ~/demo $ ./a.out
```

```
5 15
```

```
3 2 4 5 1
```

```
#move = 3
```

堆數 箱子總個數

```
Slighten@SlightenCheng ttys000-bash 06:28 ~/demo $ ./a.out
```

```
3 0
```

```
0 0 0
```

```
#move = 0
```

```
Slighten@SlightenCheng ttys000-bash 06:29 ~/demo $ ./a.out
```

```
6 12
```

```
0 0 0 0 0 12
```

```
#move = 10
```

```
Slighten@SlightenCheng ttys000-bash 06:29 ~/demo $ ./a.out
```

```
5 15
```

```
5 4 3 2 1
```

```
#move = 3
```

```
Slighten@SlightenCheng ttys000-bash 06:30 ~/demo $ ./a.out
```

```
1 7
```

```
7
```

```
#move = 0
```

```
Slighten@SlightenCheng ttys000-bash 06:34 ~/demo $ ./a.out
```

```
4 20
```

```
6 6 6 2
```

```
#move = 3
```

```
Slighten@SlightenCheng ttys000-bash 06:34 ~/demo $ ./a.out
```

```
3 15
```

```
1 2 12
```

```
#move = 7
```

```
Slighten@SlightenCheng ttys000-bash 06:34 ~/demo $ ./a.out
```

```
3 99
```

```
0 0 99
```

```
#move = 66
```

```
Slighten@SlightenCheng ttys000-bash 06:35 ~/demo $ ./a.out
```

```
2 100
```

```
1 99
```

```
#move = 49
```

```
Slighten@SlightenCheng ttys000-bash 06:35 ~/demo $ ./a.out
```

```
5 25
```

```
4 4 4 4 9
```

```
#move = 4
```

作業 Part III

Catch Pokemon! (1/2)

➤ 最近很流行 Pokémon GO (精靈寶可夢 GO)



作業 Part III

Catch Pokemon! (2/2)

- 檔名叫作 `catchpoke.cpp`
- 身為神奇寶貝訓練師又是宅宅的你，想知道如果你宅在家不動，能夠抓到幾隻神奇寶貝。假設你的探測圓周半徑為 5 單位，有無限顆寶貝球，並且只要出現在你周圍的神奇寶貝必定能抓到，不考慮時間因素。
- Input: 第一組個數字 x, y 是你的 GPS 座標位置(不考慮 z 軸)，後面 n 組數字 $x_1, y_1, x_2, y_2, \dots, x_n, y_n$ 為 n 個神奇寶貝的位置GPS 座標位置(不考慮 z 軸)
- Output: 最多能抓到的神奇寶貝數目 `#pokemon`
= ???



Hints

1. input 座標有可能不是整數 (use `double`)
2. 距離公式為歐式距離公式

$$distance = \sqrt{(x - x_i)^2 + (y - y_i)^2}$$

3. 開根號函式為 `double sqrt(double x);`
需 `#include <cmath>`

• 用法請參考：<http://www.cplusplus.com/reference/cmath/sqrt/>



Simple Testcases

你的位置

所有神奇
寶貝的位
置

```
Slighten@SlightenCheng ttys000-bash 07:21 ~/demo $ g++ catchpoke.cpp
```

```
Slighten@SlightenCheng ttys000-bash 07:21 ~/demo $ ./a.out
```

0 0

1 0

0 -1

1 -1

1 1

1 2

2 3

3 4

5 0

-5 0

4 4

4 4.5

3 4.5

6 0

100 0

3.2 4.0

在你的周
圍的神奇
寶貝

```
#pokemon = 9
```

Testcases Generator

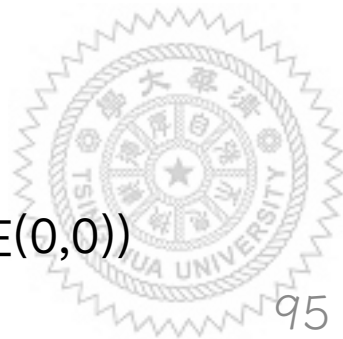


The image shows a VIM editor window titled "output.cpp + (~/.demo) - VIM". The code is written in C++ and is as follows:

```
1 #include <iostream>
2 #include <ctime>
3 #include <cstdlib>
4 using namespace std;
5
6 // 產生 -5 ~ +5 的浮點數亂數
7 double r(){
8     return ((double) rand()/RAND_MAX - 0.5) * 10;
9 }
10
11 int main(){
12     srand(time(NULL));
13     int num = rand() % 1000 + 10; // 產生 10 ~ 1010 組測資
14     while (num--){
15         cout << r() << " " << r() << endl;
16     }
17 }
```

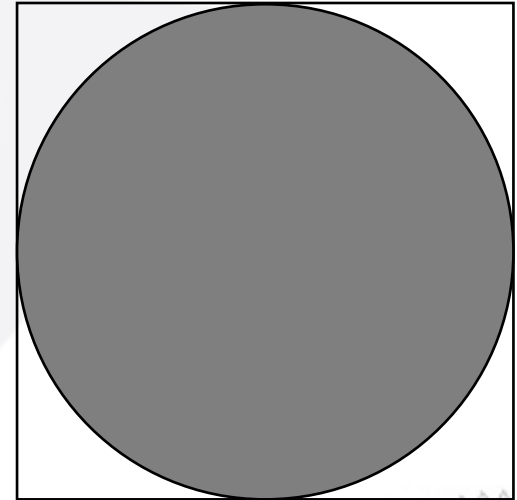
Complicated Testcases (1/2)

- input1.txt: total 80 pokemons
#pokemon = 26
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input1.txt>
- input2.txt: total 130 pokemons
#pokemon = 53
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input2.txt>
- input3.txt: total 937 pokemons
#pokemon = 279
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input3.txt>
- input4.txt: total 313 pokemons
#pokemon = 162
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input4.txt>
- input5.txt: total 663 pokemons
#pokemon = 286
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input5.txt>
- $\frac{5^2\pi}{10^2} \approx 0.785938$ (應該要很接近 $\frac{\#pokemon}{total}$ ，卻沒有？因為主角的位置不在(0,0))



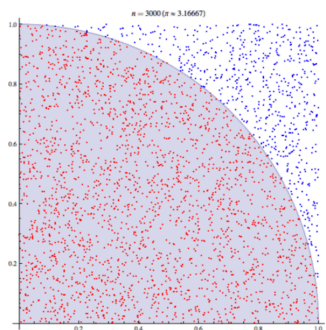
Complicated Testcases (2/2)

- $\frac{5^2\pi}{10^2} \approx 0.785938$ (應該要很接近 $\frac{\#pokemon}{total}$ ，主角的位置在(0,0))
- input6.txt: total 440 pokemons
#pokemon = 343, $\frac{343}{440} \approx 0.77954545$
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input6.txt>
- input7.txt: total 861 pokemons
#pokemon = 684, $\frac{684}{861} \approx 0.79442509$
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input7.txt>
- input8.txt: total 668 pokemons
#pokemon = 537, $\frac{537}{668} \approx 0.80389222$
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input8.txt>
- input9.txt: total 89 pokemons
#pokemon = 67, $\frac{67}{89} \approx 0.75280899$
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input9.txt>
- input10.txt: total 124 pokemons
#pokemon = 92, $\frac{92}{124} \approx 0.74193548$
 - <http://m101.nthu.edu.tw/~s101021120/Myweb/data/input10.txt>

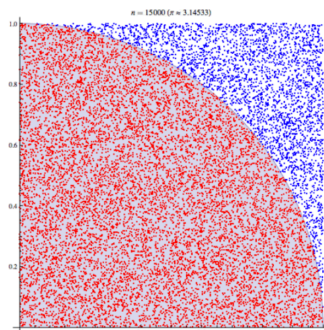


蒙地卡羅法 (Monte Carlo Method)

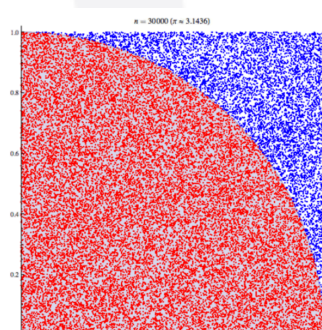
- 事實上前一頁的方法就是蒙地卡羅法求 π 的過程，將神奇寶貝的數目一直增加，最後比值會趨近於 $\frac{\pi}{4}$ (圓形與正方形面積比值)。
- Wikipedia: <https://zh.wikipedia.org/zh-tw/%E8%92%99%E5%9C%B0%E5%8D%A1%E7%BE%85%E6%96%B9%E6%B3%95?oldformat=true>



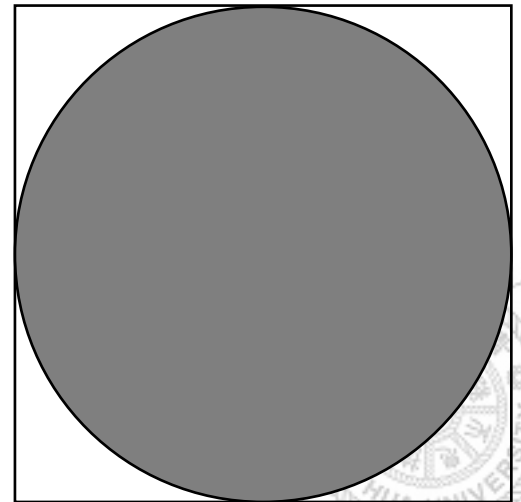
使用蒙地卡羅方法估算 π 值。放置30000個隨機點後， π 的估算值與真實值相差0.07%。



使用蒙地卡羅方法估算 π 值。放置30000個隨機點後， π 的估算值與真實值相差0.07%。



使用蒙地卡羅方法估算 π 值。放置30000個隨機點後， π 的估算值與真實值相差0.07%。



作業 Part IV:

The $3n + 1$ Problem with Exception Handling (1/2)

➤ 檔名叫作 $3n+1.cpp$

1. prompt "enter a positive integer: "
2. cin 使用者輸入 1 個數 n
3. 若 n 為正整數
 - 執行迴圈
 - 若 n 為奇數， $n = 3n+1$
 - 若 n 為偶數， $n /= 2$
 - 直到 $n == 1$ 為止
 - 輸出 n 的變化過程
4. 若 n 非正整數，請直接 output $n + \text{" is not a positive integer"}$ ，然後停止程式



作業 Part IV:

The $3n + 1$ Problem with Exception Handling (2/2)

➤ Hint:

1. 如何判斷 n 為正數？
2. 如何判斷 n 為奇數還是偶數？
3. loop 的 test 是什麼？
4. 在何時 cout?
5. You may use `/`, `%`, `cin`, `cout`
6. "while" or "for" loop?
7. "? :" or "if-else"?



3n + 1 Pseudo Code

1. input n
2. print n
3. if $n = 1$ then STOP
4. if n is odd then $n \leftarrow 3n + 1$
5. else $n \leftarrow \frac{n}{2}$
6. GOTO 2



Sample test cases

```
enter a positive integer: -3
-3 is not a positive integer
```

```
enter a positive integer: 8
8 4 2 1
```

```
enter a positive integer: 0
0 is not a positive integer
```

```
enter a positive integer: 15
15 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
```

```
enter a positive integer: 27
27 82 41 124 62 31 94 47 142 71 214 107 322 161 484 242 121 364 182 91 274 137 412 206 103 310 155 466 233 700 350 175 526 263 790 395 1186 593 1780 8
90 445 1336 668 334 167 502 251 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822 911 2734 1367
4102 2051 6154 3077 9232 4616 2308 1154 577 1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23 70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
```

```
enter a positive integer: 1
1
```

```
enter a positive integer: 22
22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

```
enter a positive integer: 3
3 10 5 16 8 4 2 1
```

```
enter a positive integer: 128
128 64 32 16 8 4 2 1
```



Mini Project: 猜數字

- 檔名叫作 `guess.cpp`
- 有兩個 player
 1. 電腦先亂數產生一個 range 0~99 之間的數字 (answer)
 2. `cout << "P1 enter the number (0~99): ";`
 3. `cin >> guess;`
 4. P1 先猜 (guess)
 - 若猜中，`cout "P1 Win!"` 然後結束遊戲。
 - guess 比 answer 小，`cout "Too small!"`，range 改成 `guess+1~99`
 - guess 比 answer 大，`cout "Too large!"`，range 改成 `0~guess-1`
- 換 P2 猜 (猜的過程與 P1 同，是否應該放在同一個迴圈?)
- 直到一方猜中或一方輸入 -1 離開遊戲為止



Sample Flow I

猜超過 range 的數字
不會影響 range

• 注意：

1. P1, P2 交換
2. range 變化
3. 大小判斷
4. 獲勝判斷

```
P1 enter the number (0~99): 5
Too small!
P2 enter the number (6~99): 72
Too large!
P1 enter the number (6~71): 54
Too large!
P2 enter the number (6~53): 58
Too large!
P1 enter the number (6~53): 40
Too large!
P2 enter the number (6~39): 4
Too small!
P1 enter the number (6~39): 18
Too small!
P2 enter the number (19~39): 25
Too small!
P1 enter the number (26~39): 31
Too small!
P2 enter the number (32~39): 35
P2 Win!
```

Sample Flow II

• 注意：

1. 離開遊戲判斷

```
P1 enter the number (0~99): 2  
Too small!  
P2 enter the number (3~99): 46  
Too small!  
P1 enter the number (47~99): -1
```



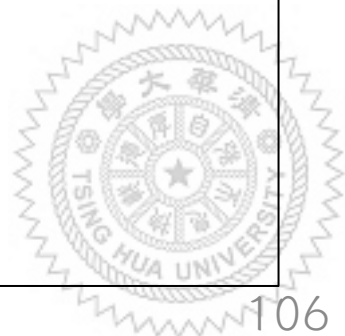
Hint

- 應先有 2 變數記住 range 的上限與下限
- 應有 1 變數存放亂數產生一個 0~99 的答案(如何產生?)
- 應有 1 變數存放使用者輸入的數字
- 應有 1 變數存放現在是Player 1 or 2
- 應有一個迴圈 (test 是?)



Template

```
#include <iostream>
#include <cstdlib> // rand(), srand()
#include <ctime>    // time()
using namespace std;
int main(){
    /* 1. declare upper, lower, answer, guess, turn (1P or 2P)
    * 2. random from 0~99, store in answer
    * 3. do loop (what is the test?):
    * 1) cout "P" << turn << "enter the number (lower~upper)\n"
    * 2) cin guess
    * 3) if (guess == ?) break;
        else if (?) cout "(P1/P2) Win!\n"
        else if (?) {cout Too small!; change range;}
        else {cout Too large!; change range;}
    4) change player (turn)
    * end loop */
}
```



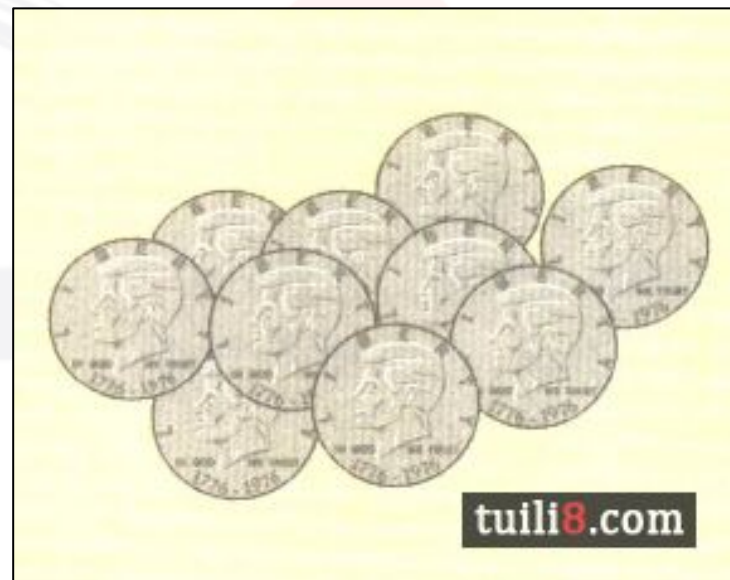
Fun Project

取硬幣賽局與其推廣



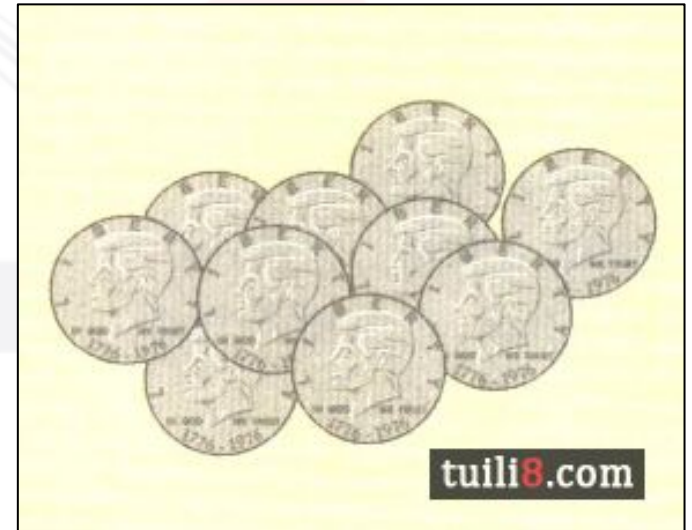
原始的取硬幣賽局

- 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



有必勝的方法。

嗯。

這個遊戲：





我全部
的策略

讓我告訴你吧



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



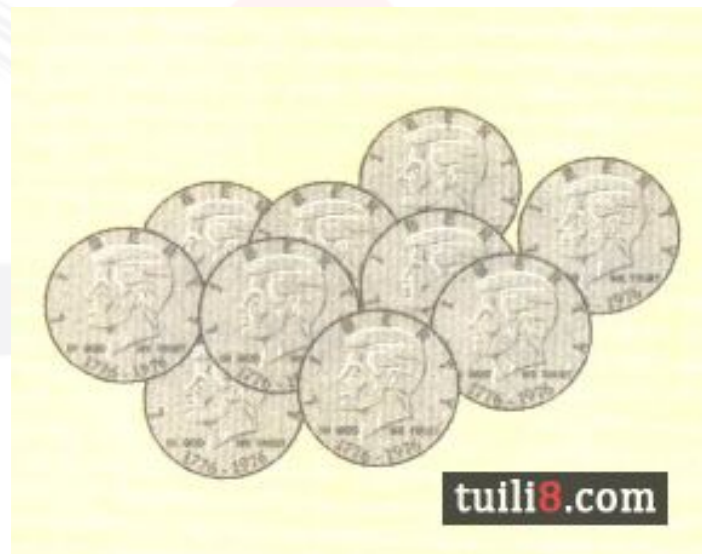
必勝法 (2/2)

- 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 $1 \sim m$ 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