# C++ Training 0319

報告人：研一 吳政彥


NTU ME Robotics Lab.
臺大機械系機器人實驗室

# Outline

- C++ Basic
- Pointer and Reference
- Class and Object
- Appendix

# C++ Basic

# Hello World!

一個C++的程式會像...

hello.cpp

```
#include<iostream>

int main(void)
{
    std::cout << "Hello World!" << std::endl;

    return 0;
}
```

引入函式庫

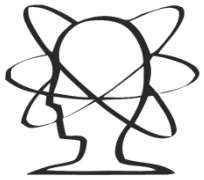
主程式進入點

程式成功結束

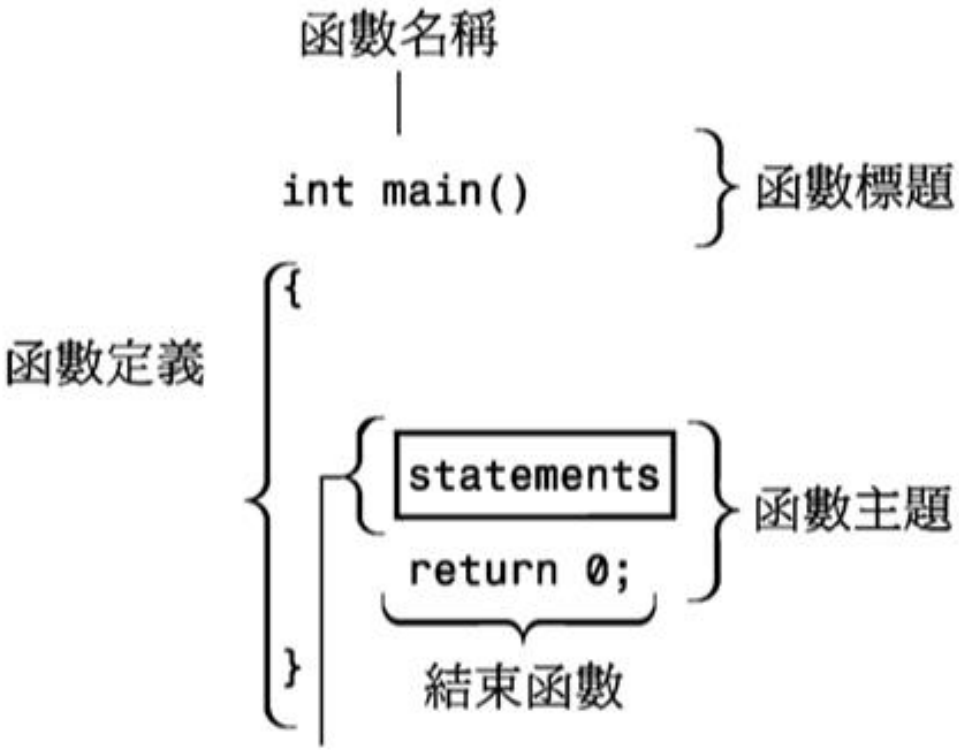
命名空間 Namespace

輸出結果

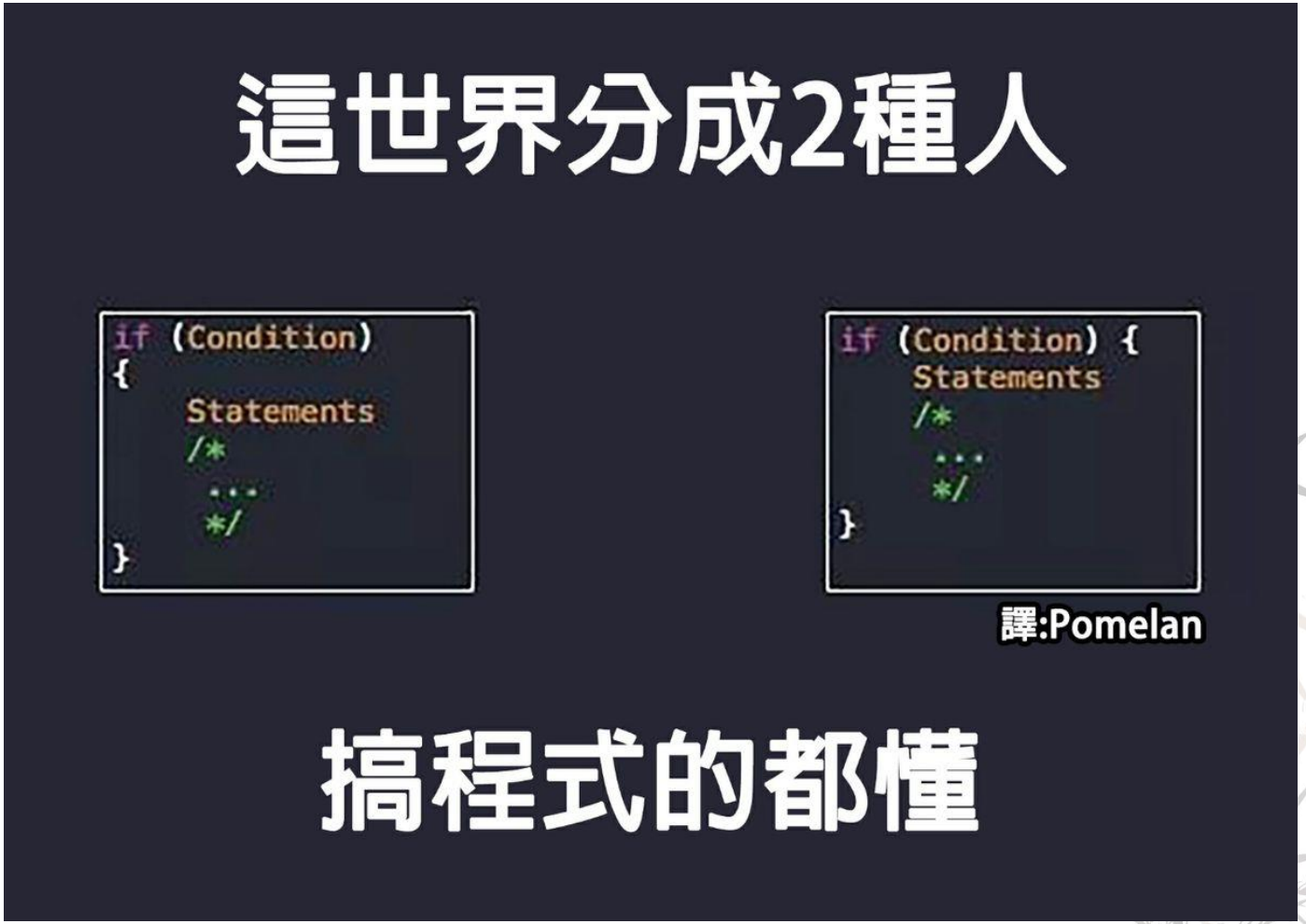
```
~/RobotLab-Cpp-Training-2022/build on main !1 ?2
./hello
Hello World!
```



# Main Function



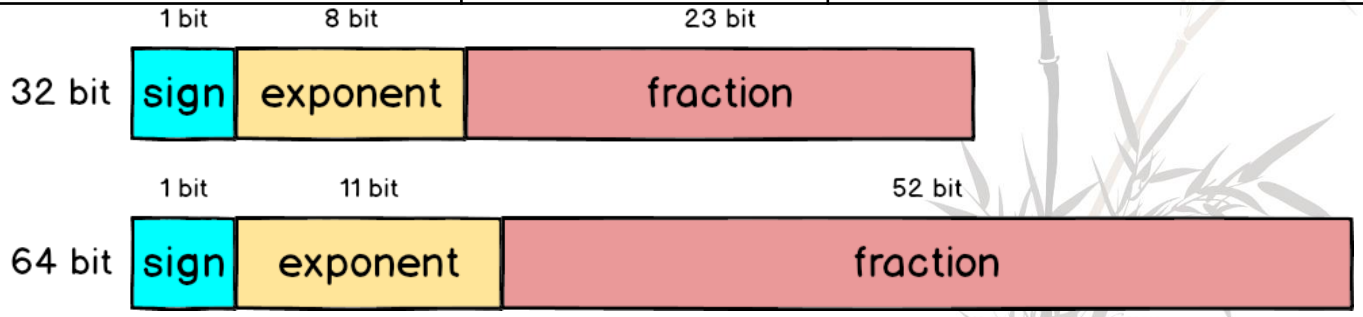
# MEME Time


這世界分成2種人
if (Condition)
{
Statements
/*
...
*/
}
if (Condition) {
Statements
/*
...
*/
}
譯:Pomelan
搞程式的都懂

# Data Type

| Data Type | Storage Type | Size | Range |
|---|---|---|---|
| int | 有號整數 | 4 bytes | -2147483648 ~ 2147483647 |
| bool | 布林值 | 1 byte | False and True |
| float | 單精度浮點數 | 4 bytes | 3.40282e+38~1.17549e-38 |
| double | 雙精度浮點數 | 8 bytes | 1.79769e+308~ 2.22507e-308 |
| long long | 有號整數 | 8 bytes | -9223372036854775808 ~ 9223372036854775807 |
| char | 字元 | 1 bytes | -127 ~ 128 |

IEEE 754



# Data Type

datatype.cpp

輸出結果

```
~/RobotLab-Cpp-Training-2022/build on main !1 ?2
./datatype
============================
int is 4 bytes.
bool is 1 bytes.
float is 4 bytes.
double is 8 bytes.
long long is 8 bytes.
char is 1 bytes.

============================
Maximum values:
int: 2147483647
bool: 1
float: 3.40282e+38
double: 1.79769e+308
long long: 9223372036854775807
char: 127

============================
Minimum values:
int: -2147483648
bool: 0
float: 1.17549e-38
double: 2.22507e-308
long long: -9223372036854775808
char: -128
```

# MEME Time

網友：為什麼中國電腦那麼慢啊？

我：因為不能用**64**位元啊！

# Input and Output (IO)

- 標準函式庫 iostream (Input and Output stream)
- 輸入 std::cin，會搭配右移運算子 >>
- 輸出 std::cout，會搭配左移運算子 <<

basicIO-1.cpp

```
#include<iostream>

using namespace std;

int main(void)
{
    int input_num;

    cin >> input_num;
    cout << "You input: " << input_num << endl;

    return 0;
}
```

# Input and Output (IO)

輸出結果

```
~/RobotLab-Cpp-Training-2022/build on main !2 ?2
./basicIO-1
45
You input: 45
```

```
~/RobotLab-Cpp-Training-2022/build on main !2 ?2
./basicIO-1
15.6
You input: 15
```

```
~/RobotLab-Cpp-Training-2022/build on main !2 ?2
./basicIO-1
A
You input: 0
```

由於定義的變數是整數，所以小數部分被捨去

std::cin當輸入不符合設定的變數型態時，會返回0

# Input and Output (IO)

- 格式化輸出入，使用iomanip (IO manipulation)函式庫
- setbase

```
// setbase
int n = 127;

cout << n << endl;
cout << "bin (base =  2): " << setbase(2) << n << endl; // It doesn't work because the standard says no
cout << "oct (base =  8): " << setbase(8) << n << endl;
cout << "dec (base = 10): " << setbase(10) << n << endl;
cout << "hex (base = 16): " << setbase(16) << n << endl;
cout << "After using setbase: " << n << endl;
cout << endl;
```

只支援8、10、16進位

basicIO-2.cpp

輸出結果

```
127
bin (base =  2): 127
oct (base =  8): 177
dec (base = 10): 127
hex (base = 16): 7f
After using setbase: 7f
```

使用iomanip的函式後會改變cout的輸出

# Input and Output (IO)

- setprecision
- fixed

basicIO-2.cpp

```
// setprecision && fixed
double PI = 3.14159265359;

cout << PI << endl;
cout << setprecision(4) << PI << endl;
cout << fixed << setprecision(4) << PI << endl;
cout << "After using setprecision && fixed: " << PI << endl;
cout << endl;
```

輸出結果

```
3.14159
3.142
3.1416
After using setprecision && fixed: 3.1416
```

使用iomanip的函式後會改變cout的輸出

# Input and Output (IO)

- setw
- setfill

basicIO-2.cpp

```
// setw && setfill
// use setbase(10) and fixed << setprecision(5) to reset cout
cout << setbase(10) << n << endl;
cout << setw(8) << n << endl;
cout << setw(8) << setfill('0') << n << endl;
cout << setw(8) << fixed << setprecision(5) << PI << endl;
cout << setw(8) << fixed << setprecision(4) << PI << endl;
```

輸出結果

```
127
     127
00000127
03.14159
003.1416
```

# Operator

- 算數運算子 +, -, *, /, %
- 指派運算子 =
- 關係運算子 >, <, ==, !=, >=, <=, etc
- 算數優先順序：由左而右依序為 (), *, /, %, +
- 指派優先順序：由右而左
- 運算子最高優先順序: ()

*注意：

整數 / 整數 → 整數 (小數無條件捨去)

整數 / 浮點數 → 浮點數

浮點數 / 整數 → 浮點數

浮點數 /浮點數 →浮點數

→ 隱式轉型

# Operator

operator-1.cpp

```
int i = 4;
int j = 3;

cout << "4 / 3 = " << i / j << endl;
cout << endl;

// implicit conversion
cout << "4 / (double)3 = " << i / (double)j << endl;
cout << "(double)4 / 3 = " << (double)i / j << endl;
cout << "(double)4 / (double)3 = " << (double)i / (double)j << endl;
cout << endl;

// explicit cast
cout << "4 / (double)3 = " << i / 3.0 << endl;
cout << "(double)4 / 3 = " << 4.0 / j << endl;
cout << endl;
```

輸出結果

```
4 / 3 = 1

4 / (double)3 = 1.33333
(double)4 / 3 = 1.33333
(double)4 / (double)3 = 1.33333

4 / (double)3 = 1.33333
(double)4 / 3 = 1.33333
```

# Operator

- i++ vs. ++i

operator-2.cpp

```cpp
#include<iostream>

using namespace std;

int main(void)
{
    int i = 0;
    cout << "i++ : " << i++ << endl;

    i = 0;
    cout << "++i : " << ++i << endl;

    return 0;
}
```

輸出結果

```
~/RobotLab-Cpp-Training-2022/build on main !5 ?3
./operator-1
i++ : 0
++i : 1
```

# Operator

- x = y = z vs. x = (y = z) vs. (x = y) = z

operator-3.cpp

```
int x, y, z;
x = 1;
y = 2;
z = 3;
x = y = z;
cout << "x = y = z" << endl;
cout << x << endl;
cout << y << endl;
cout << z << endl;
cout << endl;

x = 1;
y = 2;
z = 3;
x = (y = z);
cout << "x = (y = z)" << endl;
cout << x << endl;
cout << y << endl;
cout << z << endl;
cout << endl;

x = 1;
y = 2;
z = 3;
(x = y) = z;
cout << "(x = y) = z" << endl;
cout << x << endl;
cout << y << endl;
cout << z << endl;
cout << endl;
```

輸出結果

```
x = y = z
3
3
3

x = (y = z)
3
3
3

(x = y) = z
3
2
3
```

# If else Statement

ifelse-1.cpp

```
if(grade >= 90)
        cout << "You get A+, Great!" << endl;
    else if(grade >= 84)
        cout << "You get A, Good!" << endl;
    else if(grade >= 80)
        cout << "You get A-, Not bad!" << endl;
    else
        cout << "Who are you? You're not my child." << endl;
```

輸出結果

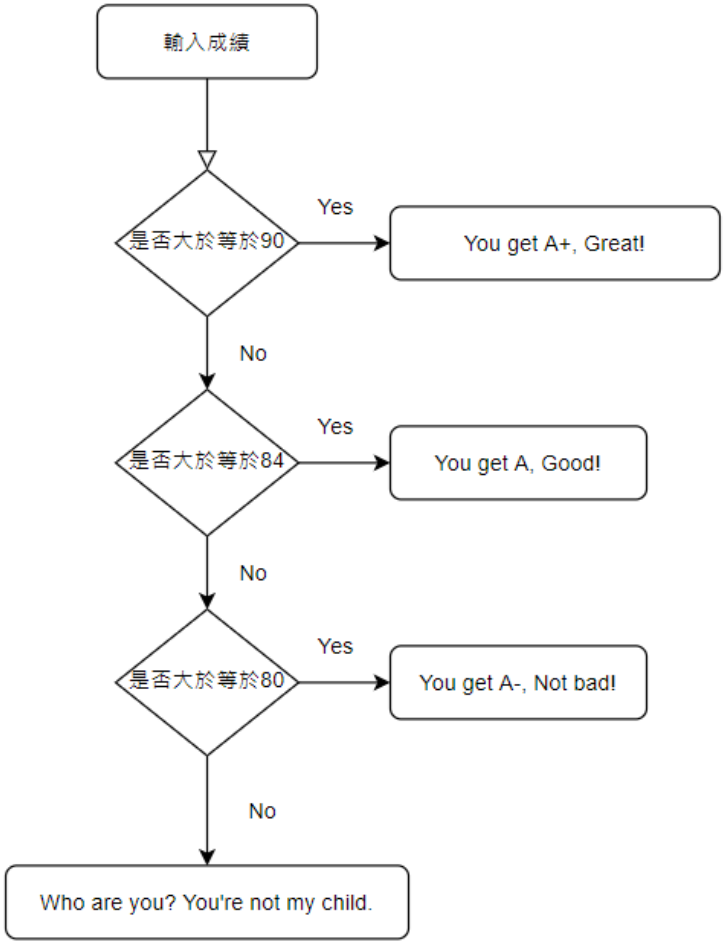
```
Please tell your grade to your Mom : 95
You get A+, Great!
```

```
Please tell your grade to your Mom : 86
You get A, Good!
```

```
Please tell your grade to your Mom : 82
You get A-, Not bad!
```

```
Please tell your grade to your Mom : 64
Who are you? You're not my child.
```

亞洲父母



# If else Statement

- 條件判斷式只能一次判斷一個條件，兩個以上要用 && (and)或者|| (or)等

ifelse-2.cpp

```
if(80 <= grade <= 100)
    cout << "You get A- ~ A+, Great!" << endl;
else
    cout << "Who are you? You're not my child." << endl;

if(80 <= grade && grade <= 100)
    cout << "You get A- ~ A+, Great!" << endl;
else
    cout << "Who are you? You're not my child." << endl;
```

輸出結果

```
Please tell your grade to your Mom : 89
You get A- ~ A+, Great!
You get A- ~ A+, Great!
```

```
Please tell your grade to your Mom : 65
You get A- ~ A+, Great!
Who are you? You're not my child.
```

# Switch Statement

switchcase-1.cpp

```
switch(grade / 10)
{
    case 10:
    case 9:
    case 8:
        cout << "A- ~ A+" << endl;
        break;
    case 7:
        cout << "B- ~ B+" << endl;
        break;
    case 6:
        cout << "C- ~ C+" << endl;
        break;
    default:
        cout << "F" << endl;
        break;
}
```

沒有break會一直做下個case

例外處理(類似else)

# Switch Statement

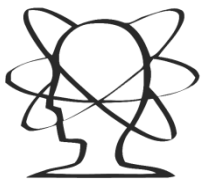
switchcase-2.cpp

```
switch(grade)
{
    case 80 ... 100:
        cout << "A- ~ A+" << endl;
        break;
    case 70 ... 79:
        cout << "B- ~ B+" << endl;
        break;
    case 60 ... 69:
        cout << "C- ~ C+" << endl;
        break;
    default:
        cout << "F" << endl;
        break;
}
```

輸出結果

```
Please input your grade : 100
A- ~ A+
```

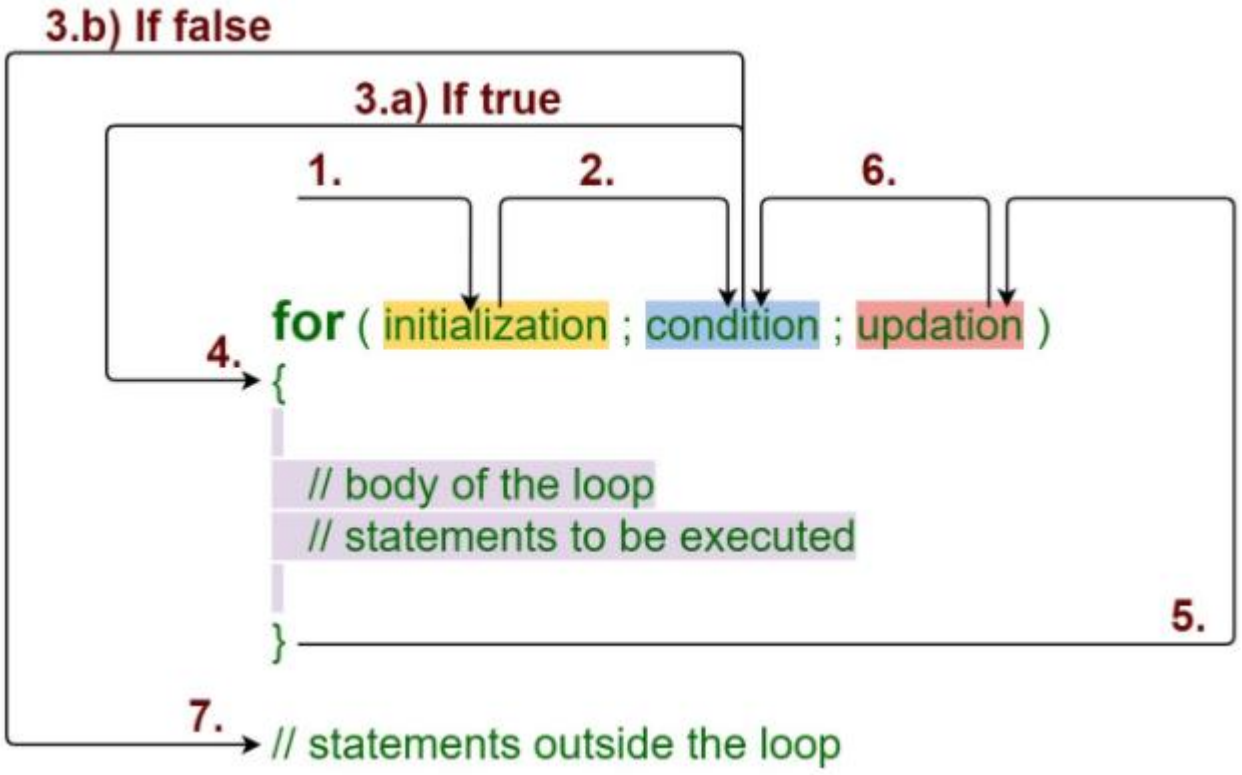
```
Please input your grade : 80
A- ~ A+
```



case中加入...，例如80...100就等於
if(grade >= 80 && grade <= 100)

# Loop

- for loop



# Loop

- for loop

forloop.cpp

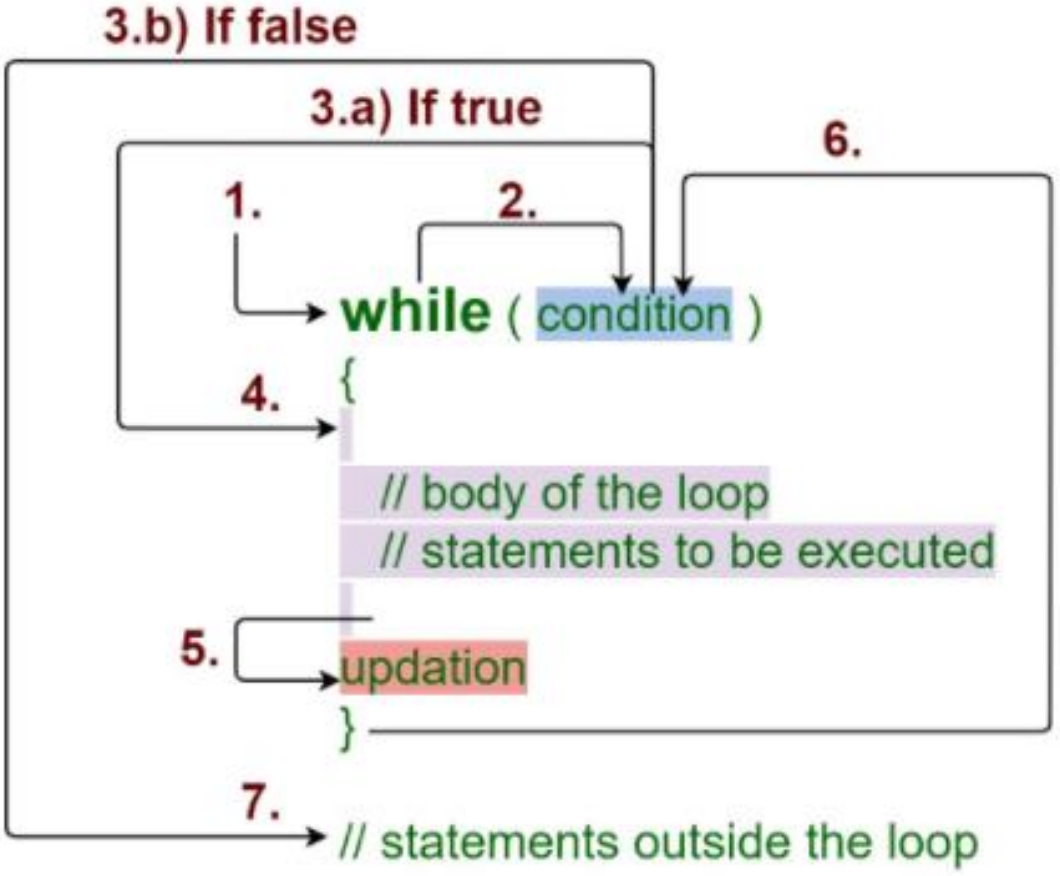
```
#include<iostream>

using namespace std;

int main(void)
{
    int i;
    int sum = 0;

    for(i = 1; i <= 100; ++i)
        sum += i;

    cout << "Result of adding 1 to 100 = " << sum << endl;
    cout << "The last i = " << i << endl;

    return 0;
}
```

輸出結果

```
Result of adding 1 to 100 = 5050
The last i = 101
```

# Loop

- while loop



# Loop

- while loop

whileloop.cpp

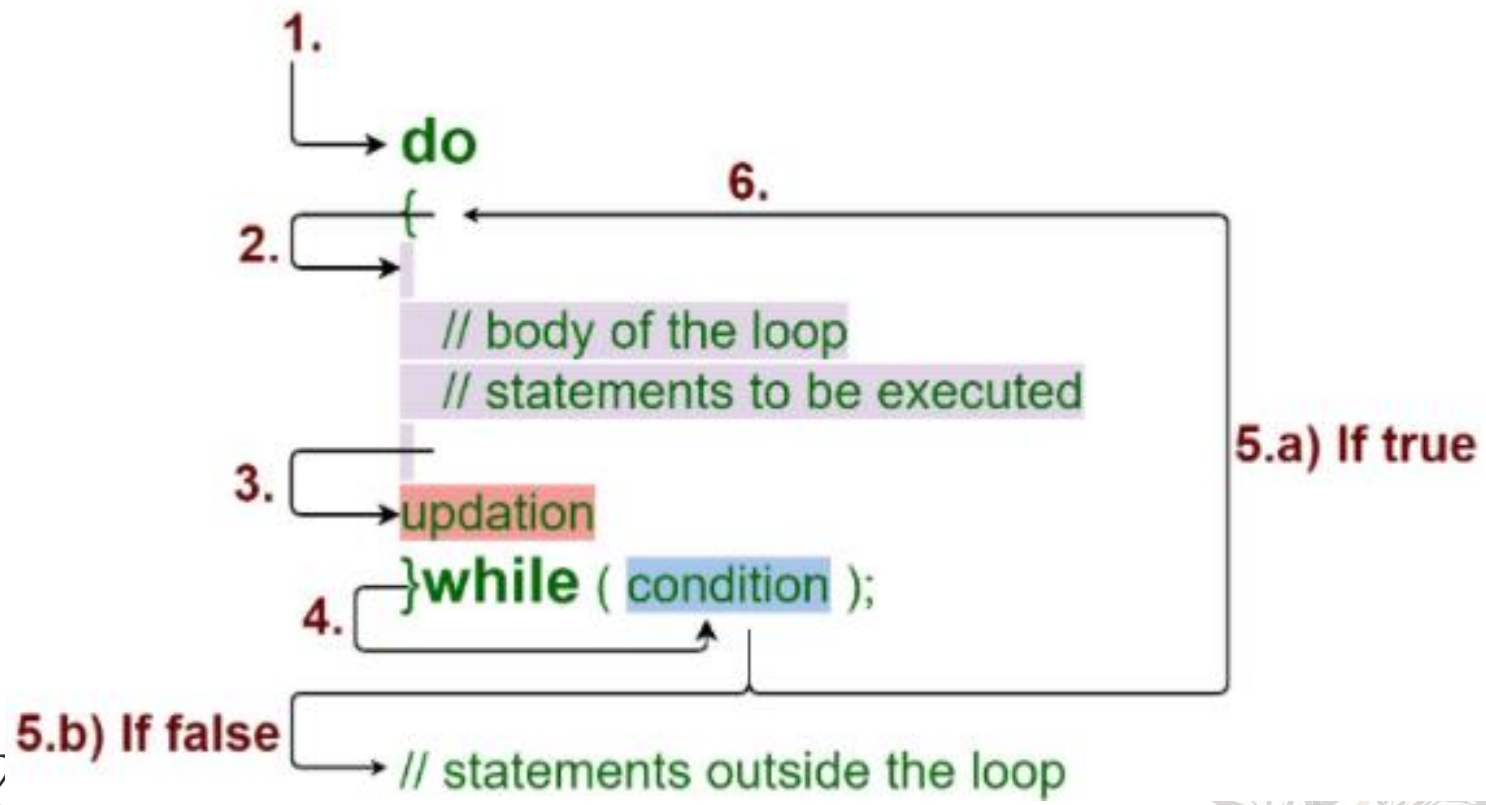
```
#include<iostream>

using namespace std;

int main(void)
{
    int i = 1;
    int sum = 0;

    while(i <= 100)
    {
        sum += i;
        ++i;
    }

    cout << "Result of adding 1 to 100 = " << sum << endl;
    cout << "The last i = " << i << endl;

    return 0;
}
```

輸出結果

```
Result of adding 1 to 100 = 5050
The last i = 101
```

# Loop

- do while loop



# Loop

- do while loop

dowhileloop.cpp

```
#include<iostream>

using namespace std;

int main(void)
{
    int i = 1;
    int sum = 0;

    do
    {
        sum += i;
        ++i;
    } while(i <= 100);

    cout << "Result of adding 1 to 100 = " << sum << endl;
    cout << "The last i = " << i << endl;

    return 0;
}
```

輸出結果

```
Result of adding 1 to 100 = 5050
The last i = 101
```

# Loop

- while loop vs. do while loop

loop.cpp

```
int i = 0;

cout << "while test" << endl;
while(i > 0 && i < 10)
{
    cout << i << " ";
    ++i;
}
cout << endl;

cout << "do while test" << endl;
do
{
    cout << i << " ";
    ++i;
} while(i > 0 && i < 10);
cout << endl;
```

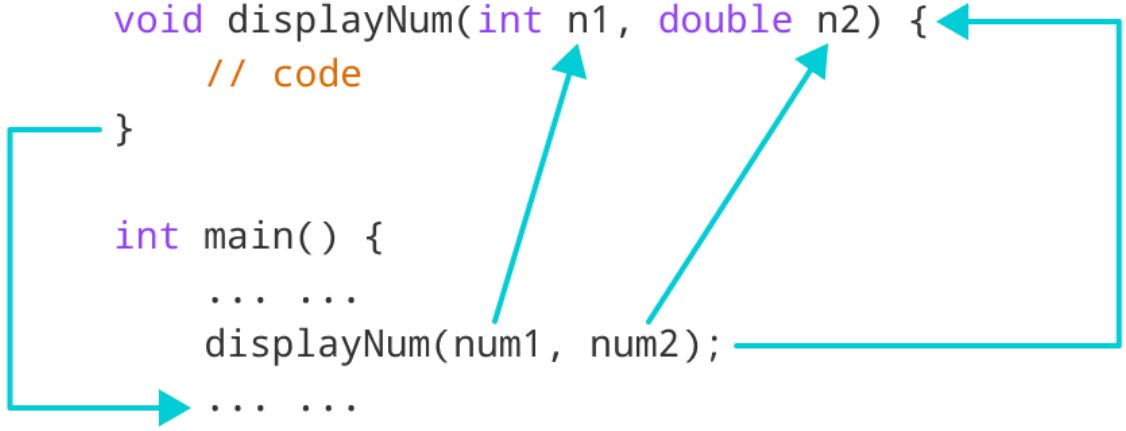
輸出結果

```
while test

do while test
0 1 2 3 4 5 6 7 8 9
```

# Function

需先設定好輸入的參數型態與回傳的型態

```
#include<iostream>

void displayNum(int n1, double n2) {
    // code
}

int main() {
    ... ...
    displayNum(num1, num2);
    ... ...
}
```



# Function

```
#include<iostream>

using namespace std;

long long factorial(int);

int main(void)
{
    int n;

    cout << "Please input an integer : ";
    cin >> n;

    cout << n << "! = " << factorial(n) << endl;

    return 0;
}

long long factorial(int n)
{
    if(n > 1)
        return (n * factorial(n-1));
    else
        return 1;
}
```

```
#include<iostream>

using namespace std;

long long factorial(int n)
{
    if(n > 1)
        return (n * factorial(n-1));
    else
        return 1;
}

int main(void)
{
    int n;

    cout << "Please input an integer : ";
    cin >> n;

    cout << n << "! = " << factorial(n) << endl;

    return 0;
}
```

# Function

- Recursion

function.cpp

```
#include<iostream>

using namespace std;

long long factorial(int);

int main(void)
{
    int n;

    cout << "Please input an integer : ";
    cin >> n;

    cout << n << "! = " << factorial(n) << endl;

    return 0;
}

long long factorial(int n)
{
    if(n > 1)
        return (n * factorial(n-1));
    else
        return 1;
}
```

輸出結果

```
Please input an integer : 10
10! = 3628800
```

# Array

array.cpp

```
cout << "Array declaration with \"int a[3];\"" << endl;
for(int i = 0; i < ARRSIZE; ++i)
    cout << a[i] << " ";

cout << endl;

int b[ARRSIZE] = {1, 2, 3};
cout << "Array declaration with \"int b[3] = {1, 2, 3};\"" << endl;
for(int i = 0; i < ARRSIZE; ++i)
    cout << b[i] << " ";

cout << endl;

int c[ARRSIZE] = {};
cout << "Array declaration with \"int c[3] = {};\"" << endl;
for(int i = 0; i < ARRSIZE; ++i)
    cout << c[i] << " ";

cout << endl;

int d[ARRSIZE] = {1};
cout << "Array declaration with \"int d[3] = {1};\"" << endl;
for(int i = 0; i < ARRSIZE; ++i)
    cout << d[i] << " ";

cout << endl;
```

輸出結果

```
Array declaration with "int a[3];"
-1870656403 21966 1780738880
Array declaration with "int b[3] = {1, 2, 3};"
1 2 3
Array declaration with "int c[3] = {};"
0 0 0
Array declaration with "int d[3] = {1};"
1 0 0
```

# MEME Time

# Namespace

- 用來防止相同輸出相同名稱函式之間的衝突

namespace.cpp

```
namespace Me
{
    // If a > b, return 1 (true)
    int compare(int a, int b)
    {
        return a > b;
    }
}

namespace You
{
    // If a < b, return 1 (true)
    int compare(int a, int b)
    {
        return a < b;
    }
}
```

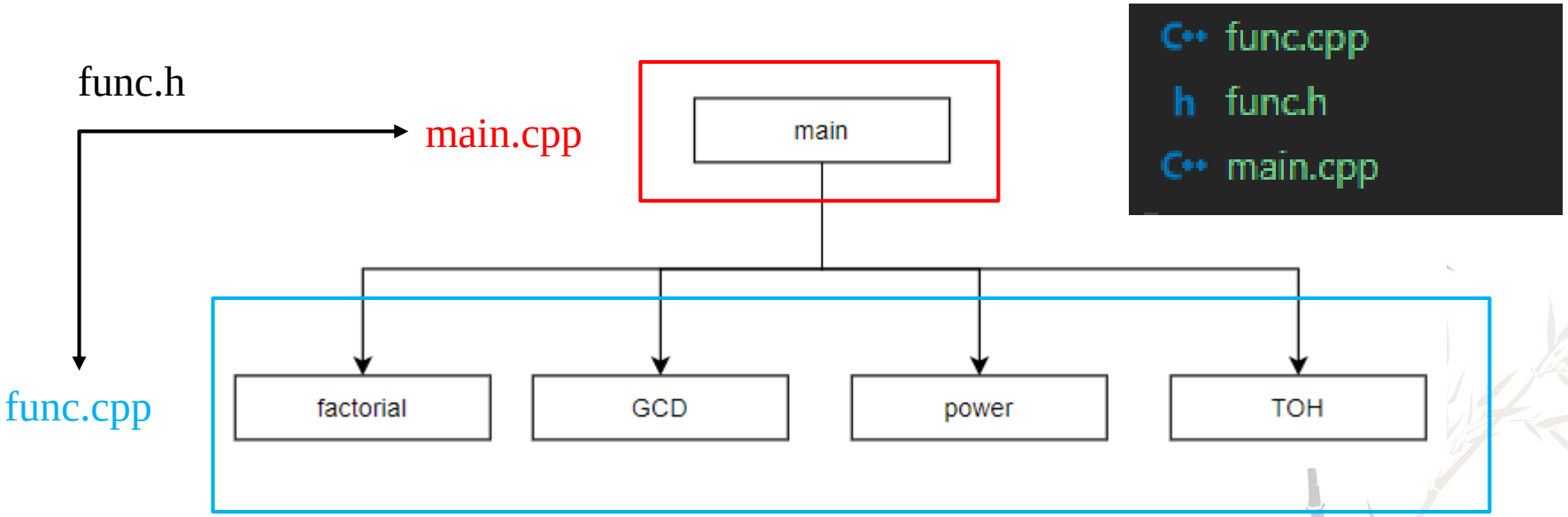
```
int a = 10;
int b = 20;

cout << "Use My namespace : " << Me::compare(a, b) << endl;
cout << "Use Your namespace : " << You::compare(a, b) << endl;
```

輸出結果

```
Use My namespace : 0
Use Your namespace : 1
```

# Header

將不同功能的函式放置其他cpp檔，並透過Header檔案來連結



# Header

func.cpp

```
#include<iostream>
#include"func.h"
```

也要include自定義的header檔案

```
long long int factorial(int n)
{
    if(n > 1)
        return (n * factorial(n-1));
    else
        return 1;
}

int GCD(int a, int b)
{
    if (a % b == 0)
        return b;
    else
        return GCD(b, a % b);
}

long long int power(int a,int b)
{
    if (b == 0)
        return 1;
    else
        return power(a, b - 1) * a;
}

void TOH(int n, char source, char target, char temp)
{
    if (n > 0)
    {
        TOH(n - 1, source, temp, target);
        std::cout << "\t" << source << " ==> "<< target << "\t\t" << n << std::endl;
        TOH(n - 1, temp, target, source);
    }
}
```

注意:
不要用全域定義的Namespace

# Header

func.h

```
#ifndef _FUNC_H_
#define _FUNC_H_

long long int factorial(int);
int GCD(int, int);
long long int power(int, int);
void TOH(int, char, char, char);

#endif
```

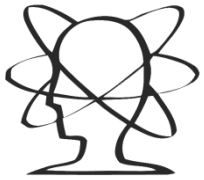
前置處理器會檢查此header檔案是否被多次呼叫
可在#define _FUNC_H_ **1**
加入數字來限制呼叫次數

# Header

main.cpp

```
#include<iostream>
#include"func.h"

using namespace std;

int main(void)
{
    int n;

    cout << "Enter number of plates : ";
    cin >> n;

    cout << "\tmove method\tdisk number" << endl;
    TOH(n, 'A', 'B', 'C');

    return 0;
}
```

自定義的header檔案要用""



# Header

輸出結果

```
Enter number of plates : 4
        move method     disk number
        A ==> C         1
        A ==> B         2
        C ==> B         1
        A ==> C         3
        B ==> A         1
        B ==> C         2
        A ==> C         1
        A ==> B         4
        C ==> B         1
        C ==> A         2
        B ==> A         1
        C ==> B         3
        A ==> C         1
        A ==> B         2
        C ==> B         1
```

# Argument

argument.cpp

```
#include<iostream>

using namespace std;

int GCD(int, int);

int main(int argc, char** argv)
{
    if(argc != 3)
    {
        cout << "Please use command " << argv[0] <<" [integer1] [integer2]" << endl;
        exit(1);
    }

    int a = atoi(argv[1]);
    int b = atoi(argv[2]);
    cout << "gcd(" << a << ", " << b << ") = " << GCD(a, b) << endl;
}

int GCD(int a, int b)
{
    if (a % b == 0)
        return b;
    else
        return GCD(b, a % b);
}
```

使用命令列引數

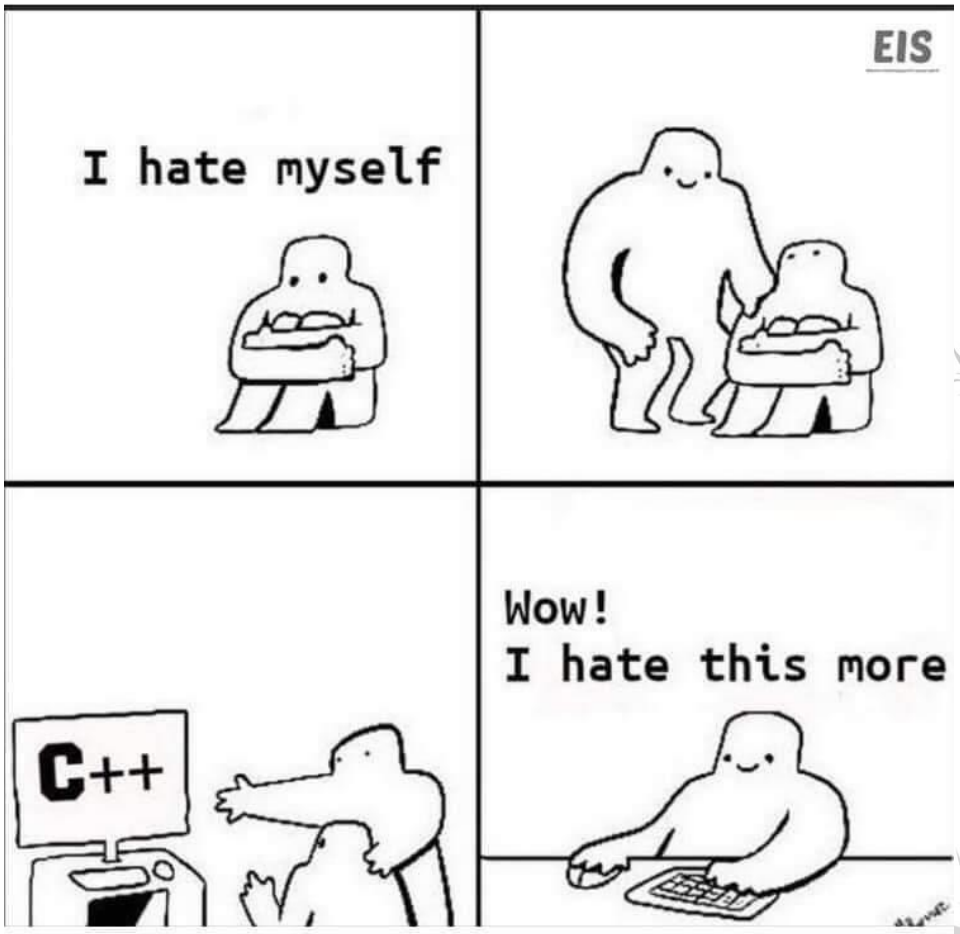
# Argument

輸出結果

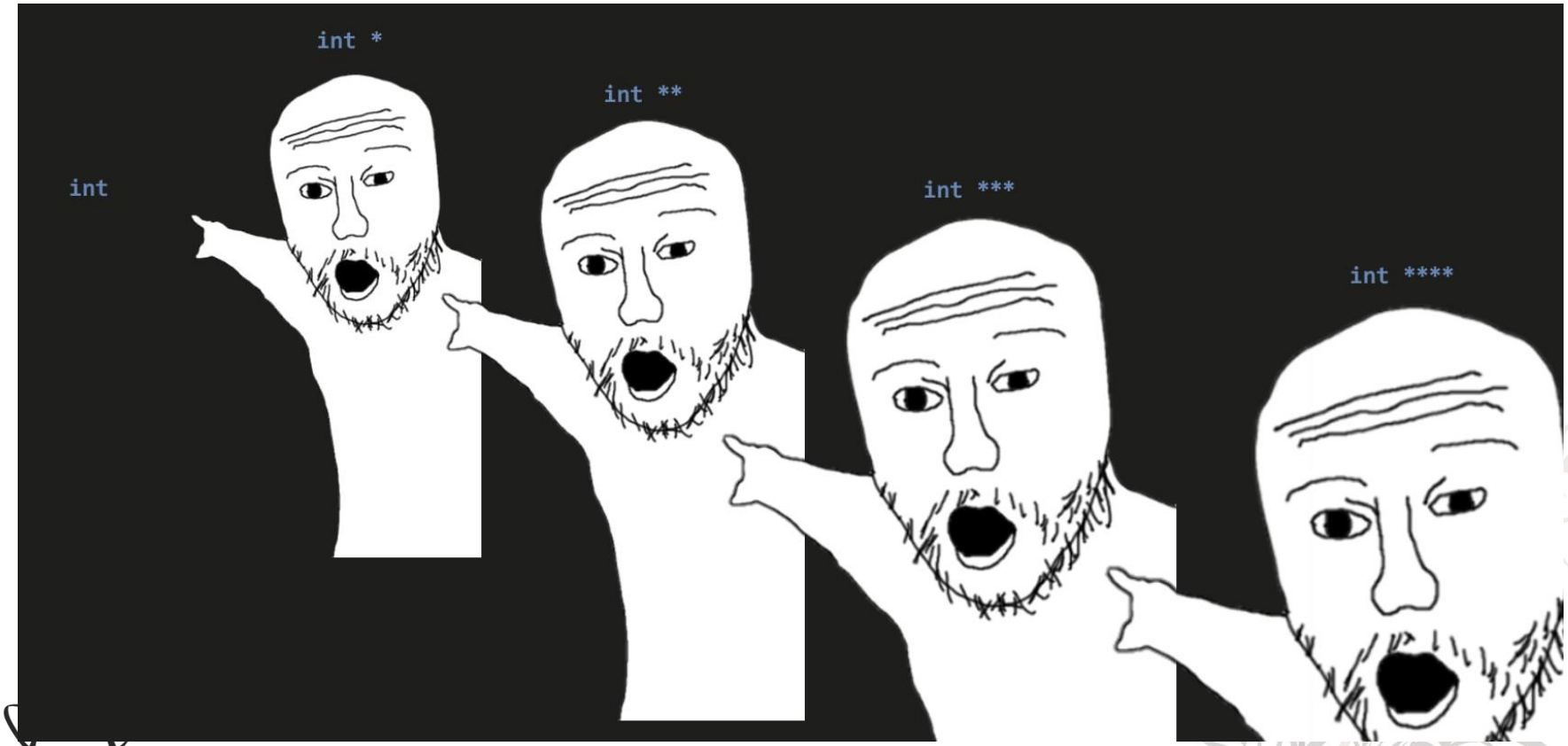
```
~/RobotLab-Cpp-Training-2022/build on main !3 ?4
 ./argument 720 56
gcd(720, 56) = 8
```

| | argc | argv[0] | argv[1] | argv[2] |
|---|---|---|---|---|
| 變數型態 | int | char* | char* | char* |
| 變數儲存狀態 | 3 | argument | 720 | 56 |

# Break Time



# Pointer and Reference



# *

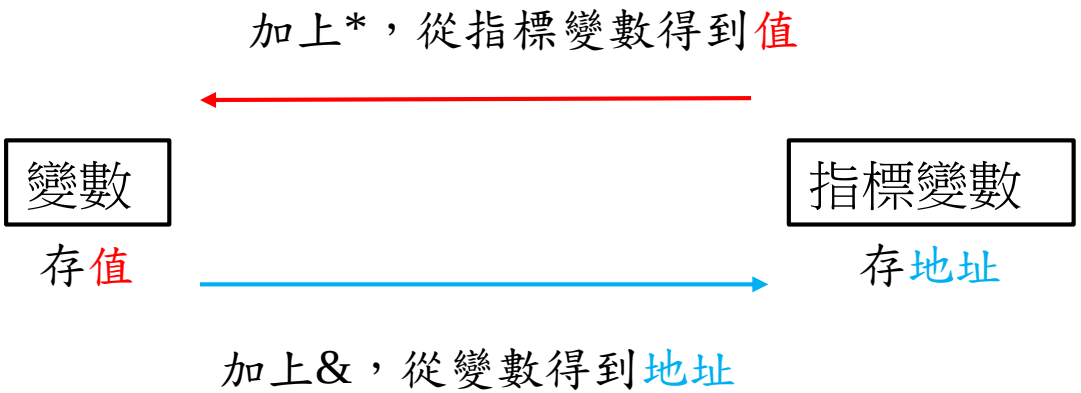
- Multiplication operator 乘法運算子
- Definition of a pointer 定義指標變數
- Dereferencing operator 提令運算子

容易搞混

```
int main()
{
    int x, y;
    int *p;

    p = &x;

    y = x * 20;
    *p = y + 100;
}
```

& : Address operator 取址運算子

# * vs. &

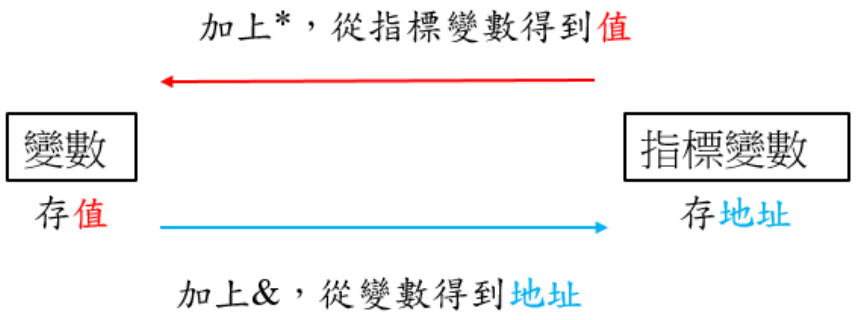


# * and &

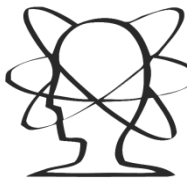
Multiplication operator 乘法運算子

Definition of a pointer 定義指標變數

Dereferencing operator 提令運算子




```
int main()
{
    int x;
    int *p = &x; // initialize p with &x
    int *q = 100; // error

    p = &x; // assign &x to p
    p = 100; // error
    *p = 100; // assign 100 to what p points
    *p = &x; // error
}
```

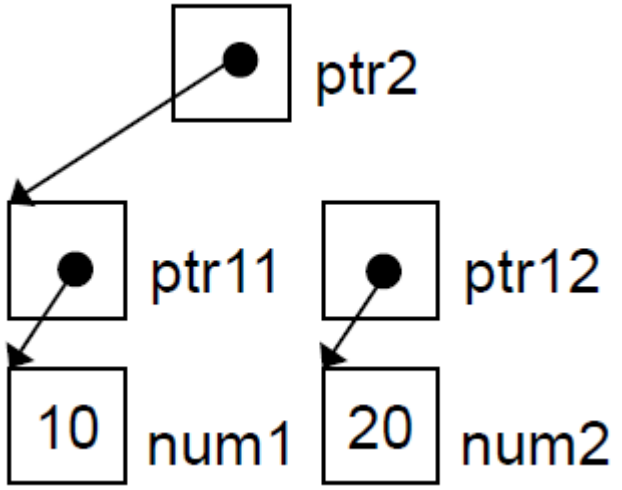


# Pointer and Individual Variables

```
int main()
{
   int num1 = 10, num2 = 20,
   *ptr11, *ptr12,
   **ptr2;

   ptr11 = &num1;
   ptr12 = &num2;
   ptr2 = &ptr11;

   *ptr11 = 30;
   *ptr2 = &num2;
   *ptr11 = 40;
   *ptr12 = 50;
   **ptr2 = 60;
}
```

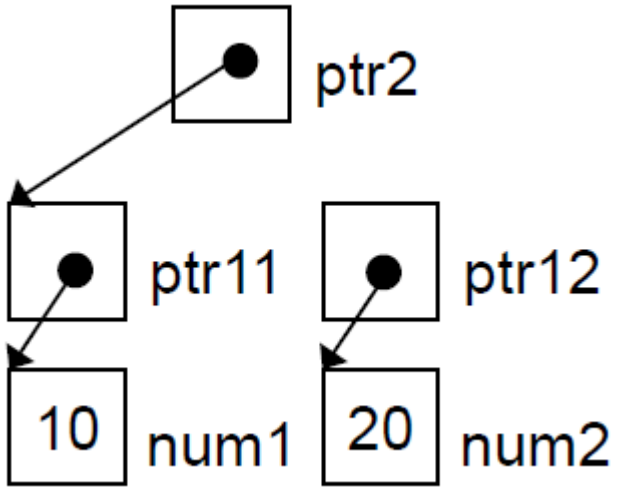
Q1：此程式中有幾個指標變數?

A: 3個



# Pointer and Individual Variables

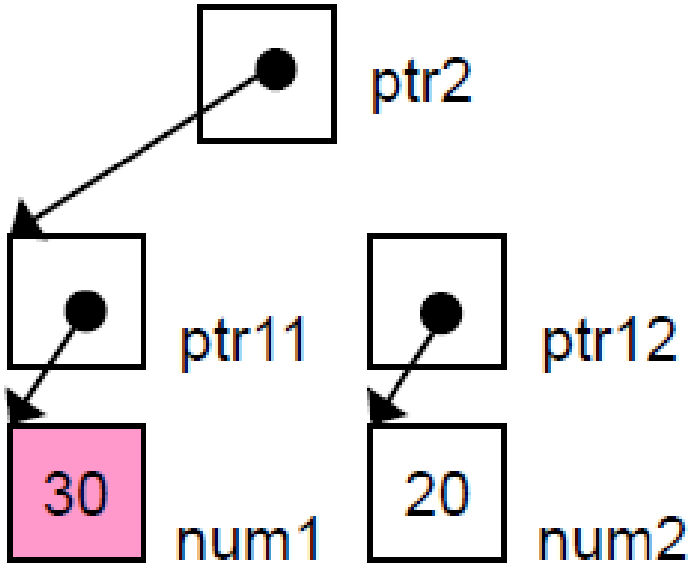
```
int main()
{
   int num1 = 10, num2 = 20,
   *ptr11, *ptr12,
   **ptr2;

   ptr11 = &num1;
   ptr12 = &num2;
   ptr2 = &ptr11;

   *ptr11 = 30;      // 1
   *ptr2 = &num2;    // 2
   *ptr11 = 40;      // 3
   *ptr12 = 50;      // 4
   **ptr2 = 60;      // 5
}
```



# Pointer and Individual Variables

```
int main()
{
   int num1 = 10, num2 = 20,
   *ptr11, *ptr12,
   **ptr2;

   ptr11 = &num1;
   ptr12 = &num2;
   ptr2 = &ptr11;

   *ptr11 = 30;      // 1
   *ptr2 = &num2;    // 2
   *ptr11 = 40;      // 3
   *ptr12 = 50;      // 4
   **ptr2 = 60;      // 5
}
```

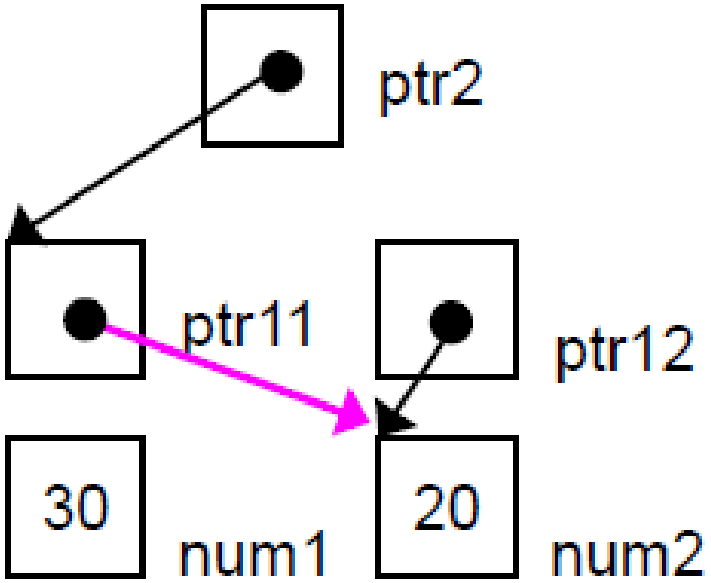
Step 1.



# Pointer and Individual Variables

```
int main()
{
   int num1 = 10, num2 = 20,
   *ptr11, *ptr12,
   **ptr2;

   ptr11 = &num1;
   ptr12 = &num2;
   ptr2 = &ptr11;

   *ptr11 = 30;      // 1
   *ptr2 = &num2;    // 2
   *ptr11 = 40;      // 3
   *ptr12 = 50;      // 4
   **ptr2 = 60;      // 5
}
```

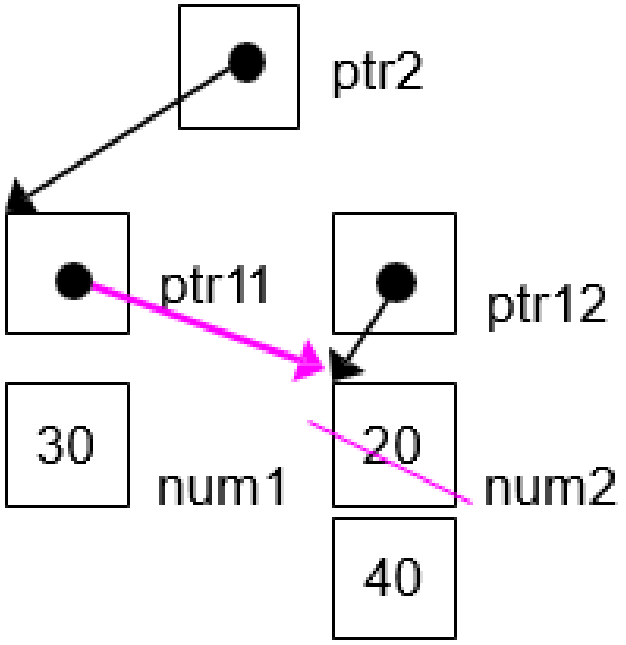
Step 2.



# Pointer and Individual Variables

```
int main()
{
   int num1 = 10, num2 = 20,
   *ptr11, *ptr12,
   **ptr2;

   ptr11 = &num1;
   ptr12 = &num2;
   ptr2 = &ptr11;

   *ptr11 = 30;     // 1
   *ptr2 = &num2;   // 2
   *ptr11 = 40;     // 3
   *ptr12 = 50;     // 4
   **ptr2 = 60;     // 5
}
```

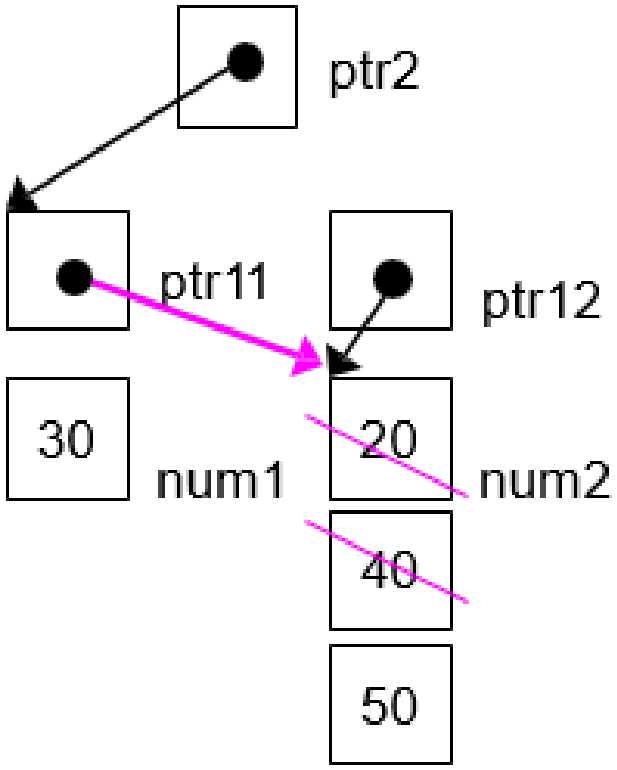
Step 3.



# Pointer and Individual Variables

```
int main()
{
   int num1 = 10, num2 = 20,
   *ptr11, *ptr12,
   **ptr2;

   ptr11 = &num1;
   ptr12 = &num2;
   ptr2 = &ptr11;

   *ptr11 = 30;      // 1
   *ptr2 = &num2;    // 2
   *ptr11 = 40;      // 3
   *ptr12 = 50;      // 4
   **ptr2 = 60;      // 5
}
```

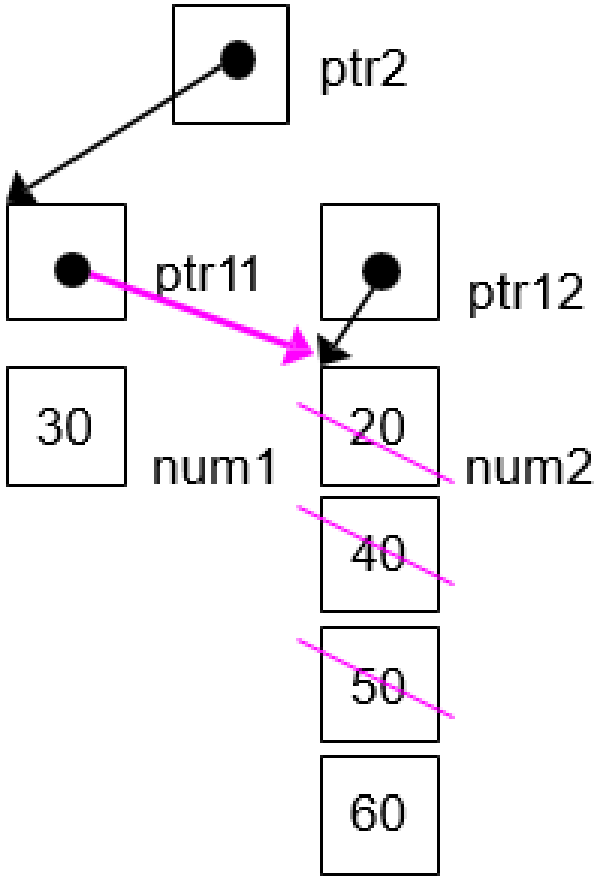
Step 4.



# Pointer and Individual Variables

```
int main()
{
   int num1 = 10, num2 = 20,
   *ptr11, *ptr12,
   **ptr2;

   ptr11 = &num1;
   ptr12 = &num2;
   ptr2 = &ptr11;

   *ptr11 = 30;      // 1
   *ptr2 = &num2;    // 2
   *ptr11 = 40;      // 3
   *ptr12 = 50;      // 4
   **ptr2 = 60;      // 5
}
```

Step 5.



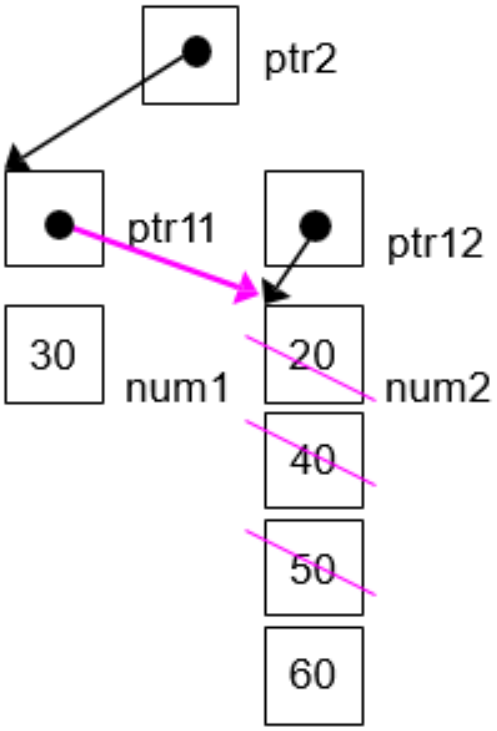
# Pointer and Individual Variables

pointer-1.cpp

```
#include<iostream>

using namespace std;

int main(void)
{
    int num1 = 10, num2 = 20,
    *ptr11, *ptr12,
    **ptr2;

    ptr11 = &num1;
    ptr12 = &num2;
    ptr2 = &ptr11;

    *ptr11 = 30;    // 1
    *ptr2 = &num2;  // 2
    *ptr11 = 40;    // 3
    *ptr12 = 50;    // 4
    **ptr2 = 60;    // 5

    cout << "num1 = " << num1 << endl;
    cout << "num2 = " << num2 << endl;

    return 0;
}
```

輸出結果

```
num1 = 30
num2 = 60
```

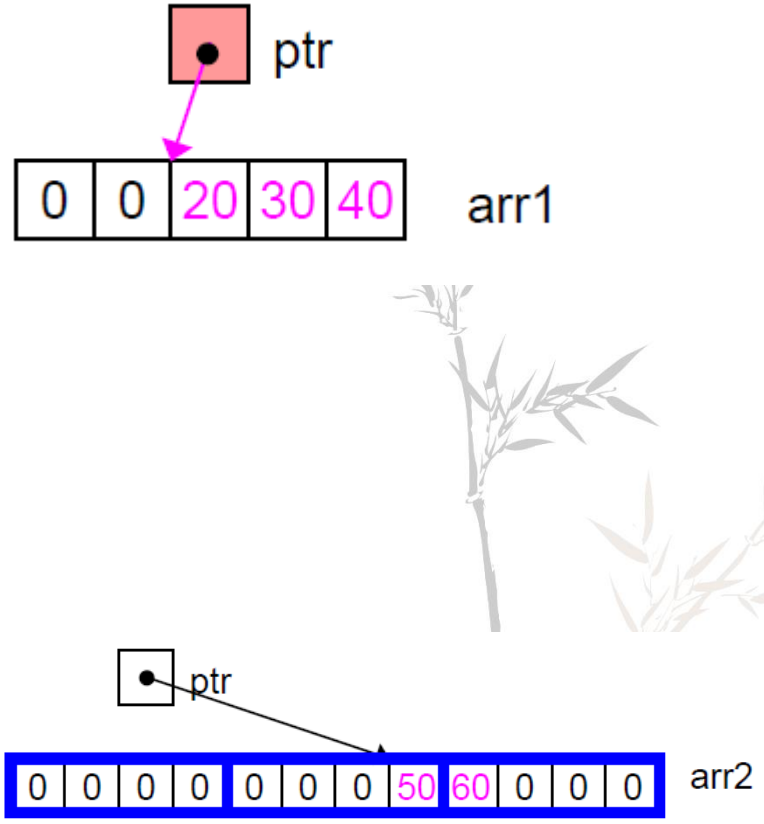


# Pointer and Array

```
#include<iostream>

using namespace std;

int main(void)
{
    int arr1[5]={}, arr2[3][4]={},
    *ptr=0;

    ptr = &arr1[2];
    *ptr = 20;
    *(ptr+1) = 30;
    ptr[2] = 40;

    ptr = &arr2[1][3];
    *ptr = 50;
    ptr[1] = 60;

    return 0;
}
```

Block 1

Block 2



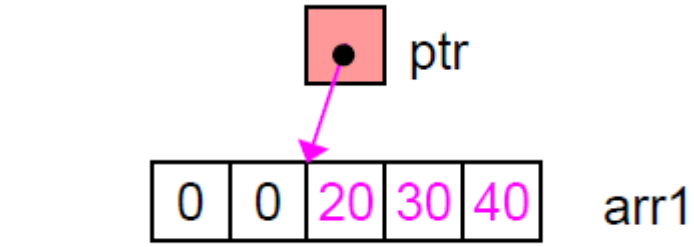
# Pointer and Array

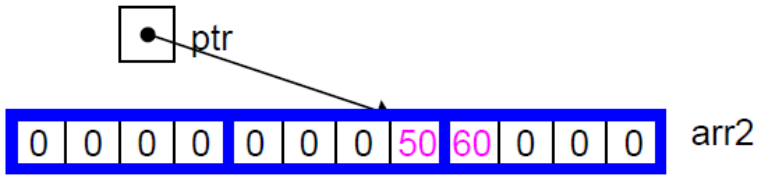
pointer-2.cpp

```
#include<iostream>

using namespace std;

int main(void)
{
    int arr1[5]={}, arr2[3][4]={},
    *ptr=0;

    ptr = &arr1[2];
    *ptr = 20;
    *(ptr+1) = 30;
    ptr[2] = 40;

    ptr = &arr2[1][3];
    *ptr = 50;
    ptr[1] = 60;

    cout << "arr1 : ";
    for(int i = 0; i < 5; ++i)
        cout << arr1[i] << " ";

    cout << endl;

    cout << "arr2 : ";
    for(int i = 0; i < 3; ++i)
    {
        for(int j = 0; j < 4; ++j)
            cout << arr2[i][j] << " ";

        cout << "| ";
    }

    cout << endl;

    return 0;
}
```

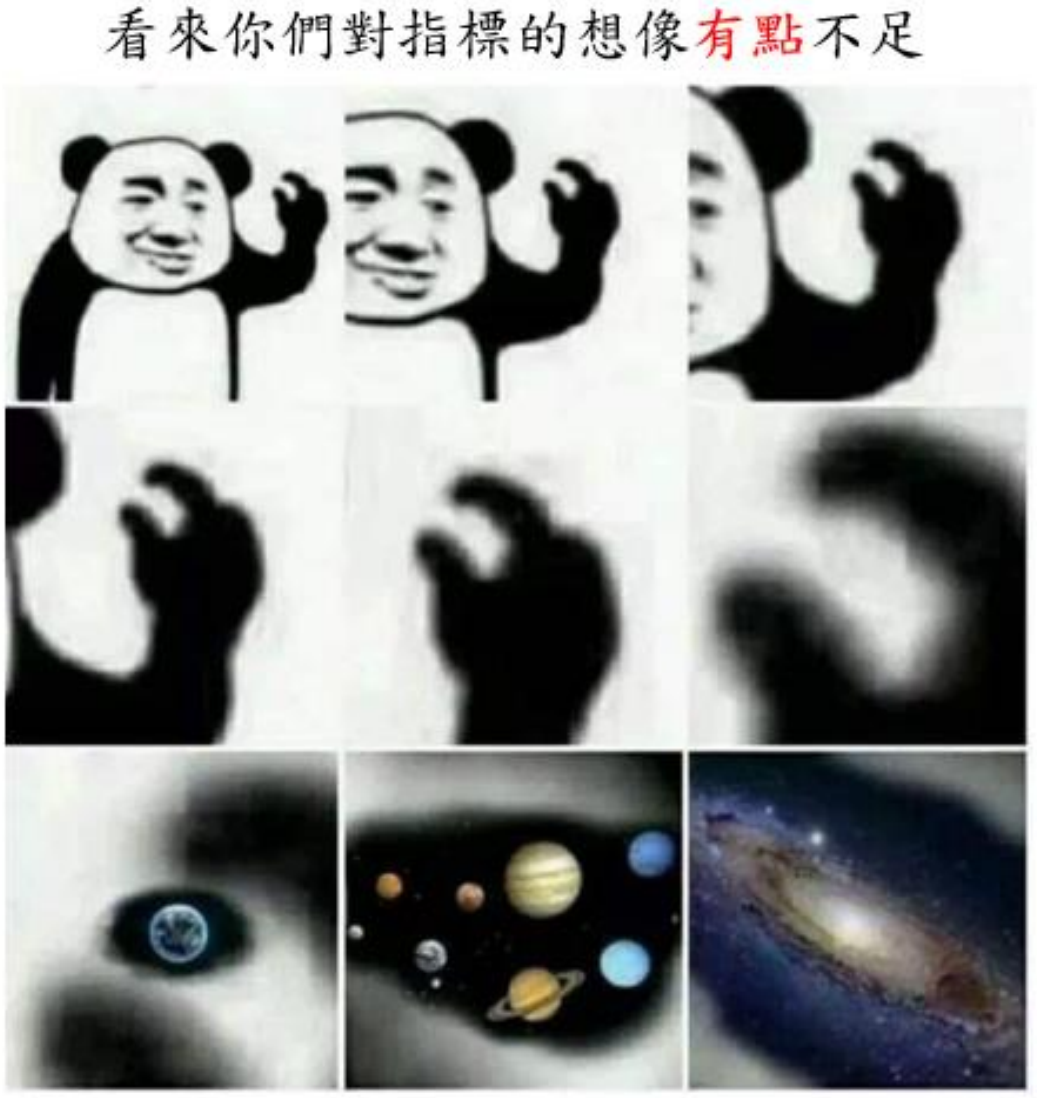
輸出結果

```
arr1 : 0 0 20 30 40
arr2 : 0 0 0 0 | 0 0 0 50 | 60 0 0 0 |
```



# MEME Time



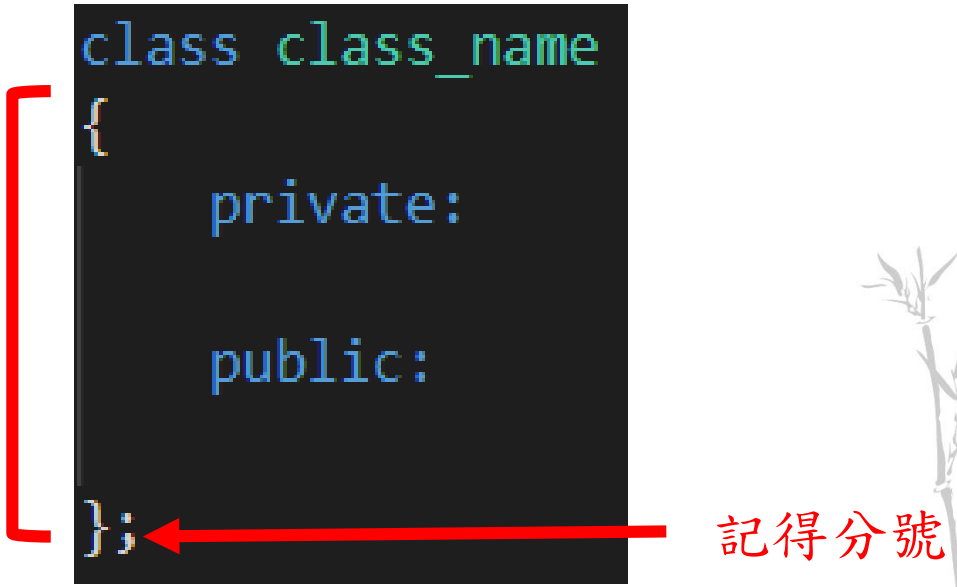
# Break Time

A code in C++

The equivalent in Python

# Class and Object

# 類別 (Class) 語法

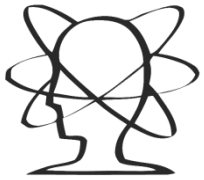



成員可以分為以下兩種:

1. **資料成員 (Data member)**
2. **成員函式 (Member function)**

# 類別成員存取限制

- public
  - 允許外界存取，作為類別的介面
- private
  - 不允許外界存取
- protected
  - 只允許衍生類別使用



# 類別建構與解構

- 類別建構函式 (Class constructor)

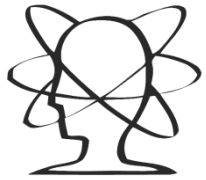
  類別產生時，會呼叫此函式來做類別初始化，此函式名稱需與類別同名稱，由於無回傳值，所以不需要宣告型態void。

- 類別解構函式 (Class destructor)

  當建構函式產生類別時，程式會自動追蹤此物件直到它消失，此時程式會自動呼叫解構函式清理此物件，此函式名稱為物件名稱前加~。靜態生成的物件會自動清理，動態生成(new)的物件需使用delete來清理。對於類別內有靜態生成的成員通常解構函式無特別任務且編譯程式會提供，所以可寫空函式或者不寫。對於類別內有動態生成的成員須明確定義，需用delete來清理。

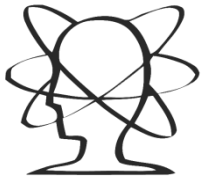
# Example – Table Tennis Player

- 建立桌球選手的類別
  - 資料成員
    - 選手的姓氏
    - 選手的名字
    - 選手是否有球桌
  - 成員函式
    - 顯示選手名字
    - 顯示選手是否有球桌
    - 更改選手是否有球桌的資訊



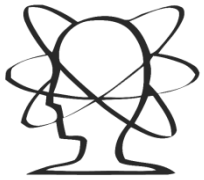
# Example - tabtenn0.h

```
class TableTennisPlayer
{
    private:
        string firstname;
        string lastname;
        bool hasTable;
    public:
        // constructor
        TableTennisPlayer(const string& fn = "none",
                          const string& ln = "none",
                          bool ht = false);

        void Name() const;
        bool HasTable() const { return hasTable; };
        void ResetTable(bool v) { hasTable = v; };

        // destructor
        ~TableTennisPlayer();
};
```



# Example - tabtenn0.cpp

```
TableTennisPlayer::TableTennisPlayer(const string& fn, const string& ln, bool ht)
{
    firstname = fn;
    lastname = ln;
    hasTable = ht;
}

void TableTennisPlayer::Name() const
{
    cout << lastname << ", " << firstname;
}

TableTennisPlayer::~TableTennisPlayer() {}
```



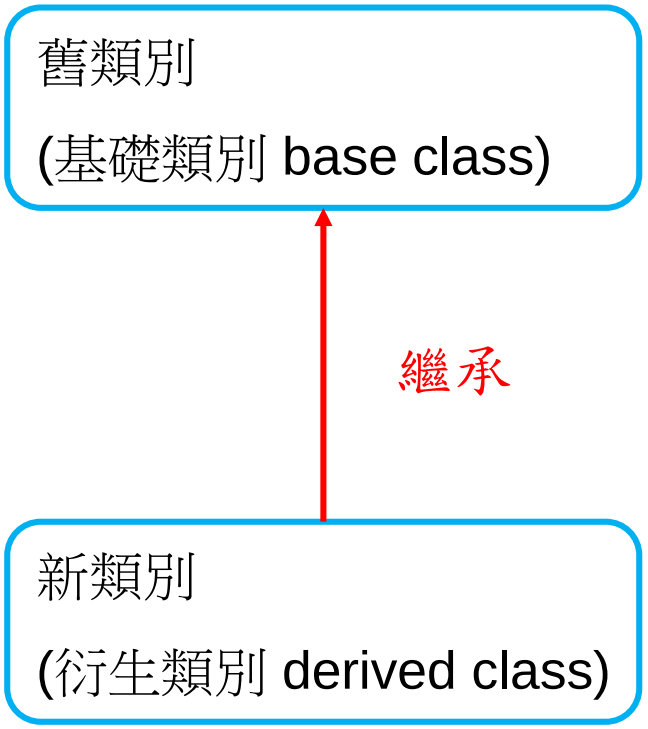
# Example – usett0.cpp

```cpp
int main(void)
{
    TableTennisPlayer player1("Chuck", "Blizzard", true);
    TableTennisPlayer player2("Tara", "Boomdea", false);

    player1.Name();
    if (player1.HasTable())
        cout << ": has a table.\n";
    else
        cout << ": hasn't a table.\n";

    player2.Name();
    if (player2.HasTable())
        cout << ": has a table.\n";
    else
        cout << ": hasn't a table.\n";

    return 0;
}
```

```
gitpod /workspace/Cplusplus-Primer-plus-writeup/Chapter13 $ ./usett0
Blizzard, Chuck: has a table.
Boomdea, Tara: hasn't a table.
```

# 繼承 Inheritance

- 目的：從舊類別中衍生出新類別



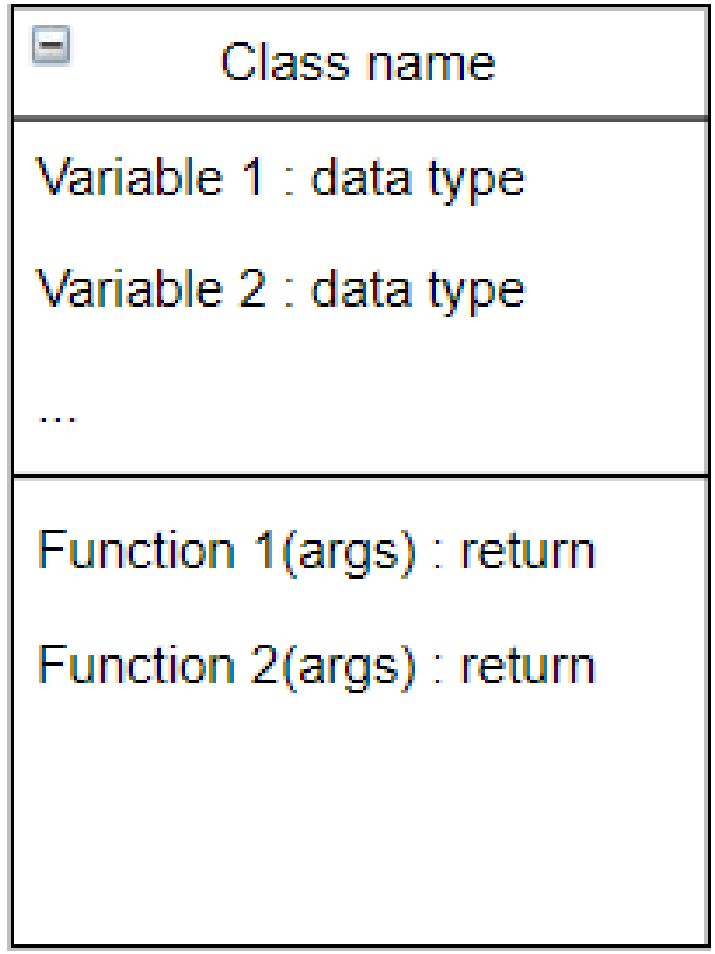

1. 增加新功能
2. 增加新的資料
3. 修改舊類別中成員函式的行為

# UML 統一塑模語言

+ : public 公用

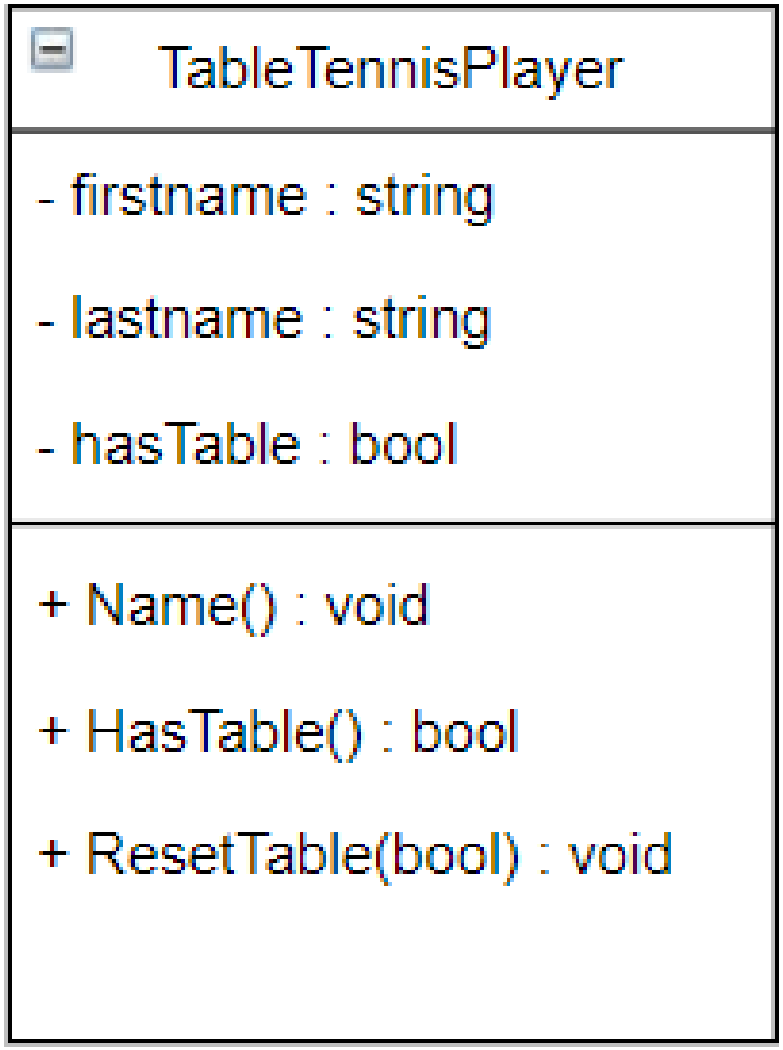
- : private 私有

# : protected 保護



# Example – tabtenn1.h

```
class TableTennisPlayer
{
private:
    string firstname;
    string lastname;
    bool hasTable;
public:
    TableTennisPlayer(const string& fn = "none",
        const string& ln = "none",
        bool ht = false);

    void Name() const;
    bool HasTable() const { return hasTable; };
    void ResetTable(bool v) { hasTable = v; };
};
```

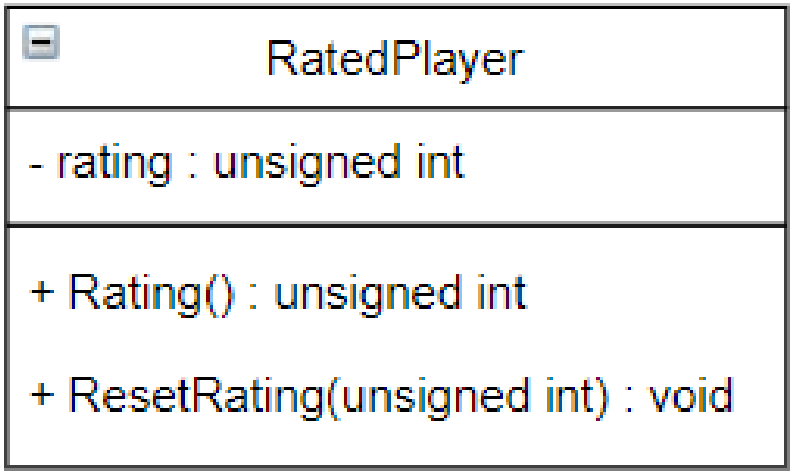


# Example – Rank Table Tennis Player

- 建立桌球選手的類別
  - 資料成員
    - 選手的姓氏
    - 選手的名字
    - 選手是否有球桌
    - 比賽積分
  - 成員函式
    - 顯示選手名字
    - 顯示選手是否有球桌
    - 更改選手是否有球桌的資訊
    - 顯示選手比賽積分
    - 更改選手比賽積分的資訊

# Example – tabtenn1.h

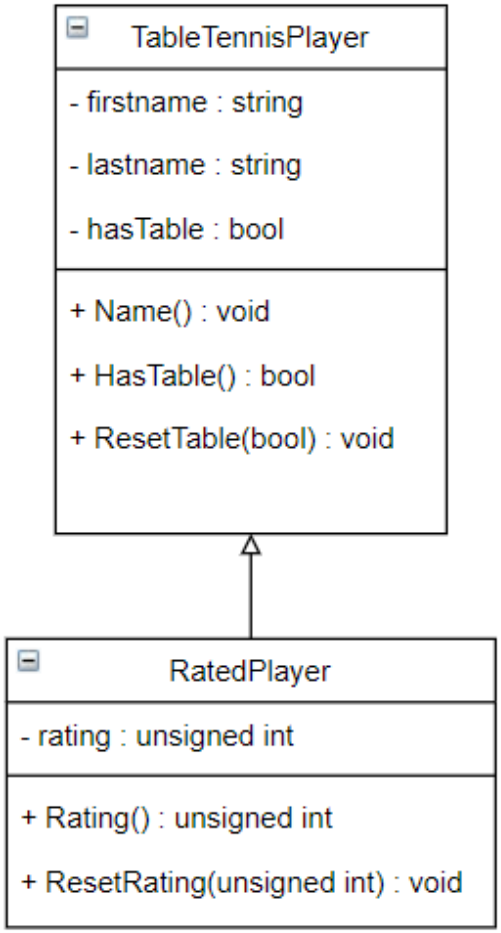
```
class RatedPlayer : public TableTennisPlayer
{
private:
    unsigned int rating;
public:
    RatedPlayer(unsigned int r = 0,
        const string& fn = "none",
        const string& ln = "none",
        bool ht = false);

    RatedPlayer(unsigned int r,
        const TableTennisPlayer& tp);

    unsigned int Rating() const { return rating; }
    void ResetRating(unsigned int r) { rating = r; }
};
```



# 繼承 – UML表示

```
class RatedPlayer : public TableTennisPlayer
{
private:
    unsigned int rating;
public:
    RatedPlayer(unsigned int r = 0,
        const string& fn = "none",
        const string& ln = "none",
        bool ht = false);

    RatedPlayer(unsigned int r,
        const TableTennisPlayer& tp);

    unsigned int Rating() const { return rating; }
    void ResetRating(unsigned int r) { rating = r; }
};
```

C++ 繼承
衍生類別：基本類別



# 衍生類別RatedPlayer

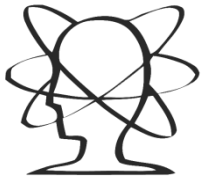
- RatedPlayer繼承TableTennisPlayer
  - RatedPlayer會包含TableTennisPlayer的資料成員
  - RatedPlayer可以使用TableTennisPlayer的公有函式
- RatedPlayer需加入自己的建構函式與額外需要的資料成員或函式

# RatedPlayer建構方法

```
class RatedPlayer : public TableTennisPlayer
{
private:
    unsigned int rating;
public:
    RatedPlayer(unsigned int r = 0,
        const string& fn = "none",
        const string& ln = "none",
        bool ht = false);

    RatedPlayer(unsigned int r,
        const TableTennisPlayer& tp);

    unsigned int Rating() const { return rating; }
    void ResetRating(unsigned int r) { rating = r; }
};
```

方法1

方法2



# Example – usettl.cpp

```
int main(void)
{
    TableTennisPlayer player1("Tara", "Boomdea", false);
    RatedPlayer rplayer1(1140, "Mallory", "Duck", true);

    rplayer1.Name();
    if (rplayer1.HasTable())
        cout << ": has a table.\n";
    else
        cout << ": hasn't a table.\n";

    player1.Name();
    if (player1.HasTable())
        cout << ": has a table.\n";
    else
        cout << ": hasn't a table.\n";

    cout << "Name: ";
    rplayer1.Name();
    cout << "; Rating: " << rplayer1.Rating() << endl;

    RatedPlayer rplayer2(1212, player1);
    cout << "Name: ";
    rplayer2.Name();
    cout << "; Rating: " << rplayer2.Rating() << endl;

    return 0;
}
```

基礎類別的建構

衍生類別的建構－方法1

衍生類別的建構－方法2

# usettl.cpp 執行結果

參數

```
TableTennisPlayer player1("Tara", "Boomdea", false);

RatedPlayer rplayer1(1140, "Mallory", "Duck", true);

RatedPlayer rplayer2(1212, player1);
```

執行結果

```
gitpod /workspace/Cplusplus-Primer-plus-writeup/Chapter13 $ ./usett1
Duck, Mallory: has a table.
Boomdea, Tara: hasn't a table.
Name: Duck, Mallory; Rating: 1140
Name: Boomdea, Tara; Rating: 1212
```

# C++函式多載

```
void func(int, int);
void func(int);

int main(void)
{
    int a = 1;
    int b = 2;

    func(1, 2);
    func(1);

    return 0;
}

void func(int n, int m)
{
    cout << "Two args : " << n << " " << m << endl;
}

void func(int n)
{
    cout << "One arg : "<< n << endl;
}
```

執行結果

```
Two args : 1 2
One arg : 1
```

# C沒有函式多載的特性

```c
void func(int, int);
void func(int);

int main(void)
{
    int a = 1;
    int b = 2;

    func(1, 2);
    func(1);

    return 0;
}

void func(int n, int m)
{
    printf("Two args : %d %d\n", n, m);
}

void func(int n)
{
    printf("One arg : %d\n", n);
}
```

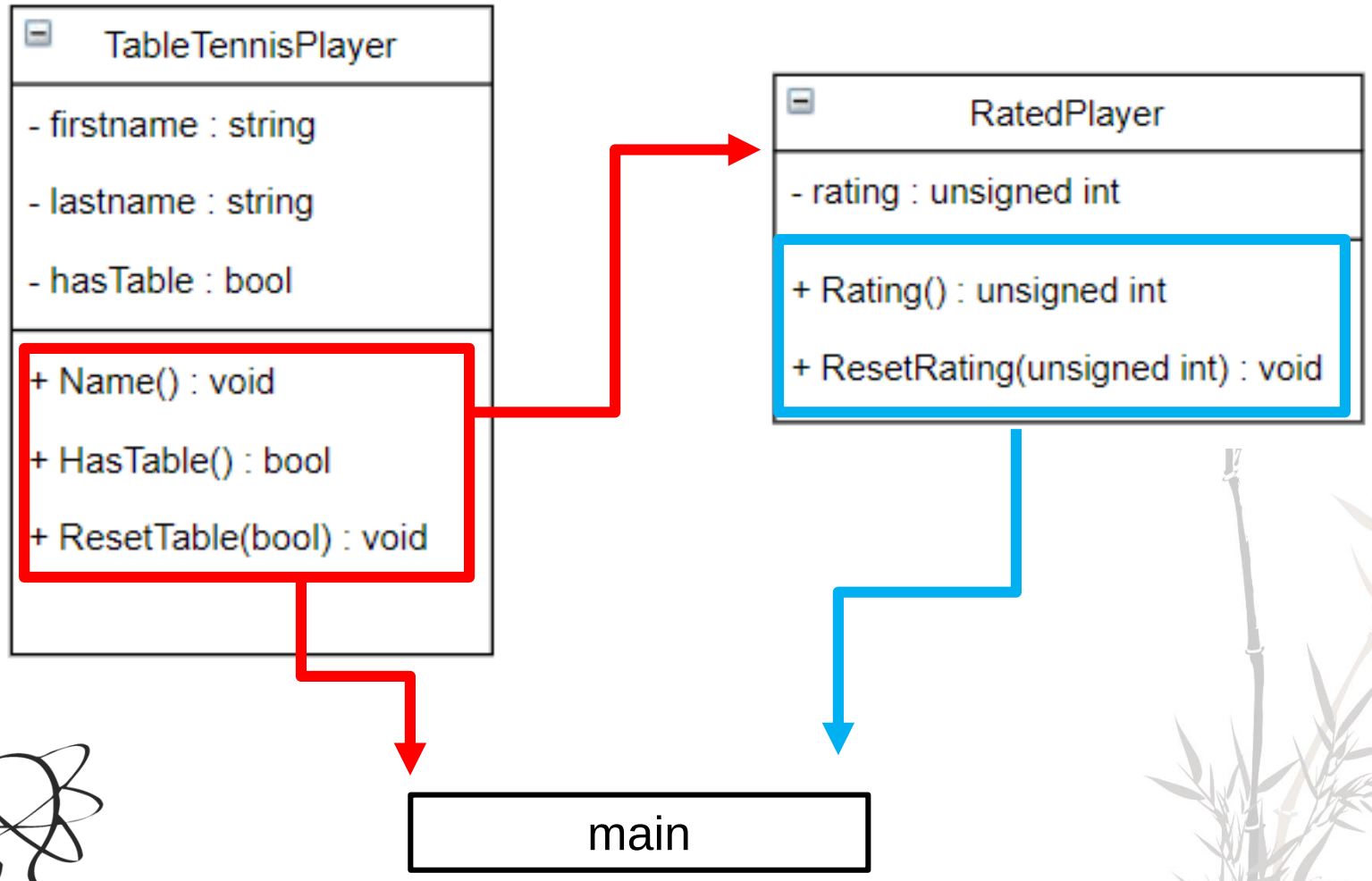
執行結果

```
1.c:4:6: error: conflicting types for ‘func’
    4 | void func(int);
      |      ^~~~
1.c:3:6: note: previous declaration of ‘func’ was here
    3 | void func(int, int);
      |      ^~~~
1.c: In function ‘main’:
1.c:11:2: error: too many arguments to function ‘func’
   11 |  func(1, 2);
      |  ^~~~
1.c:4:6: note: declared here
    4 | void func(int);
      |      ^~~~
1.c: At top level:
1.c:22:6: error: conflicting types for ‘func’
   22 | void func(int n)
      |      ^~~~
1.c:17:6: note: previous definition of ‘func’ was here
   17 | void func(int n, int m)
      |      ^~~~
```

# 類別成員存取限制

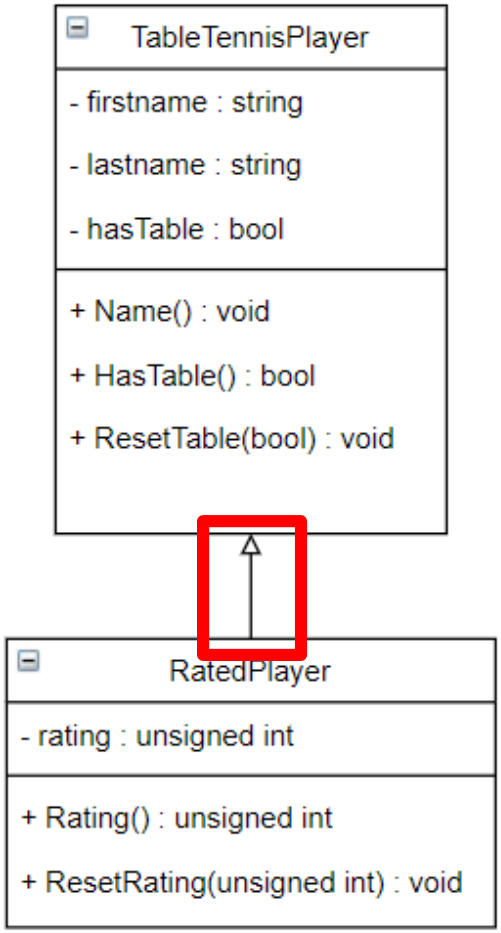
| 基礎類別的成員屬性 | 基礎類別中 | 衍生類別中 | 其他地方 |
|---|---|---|---|
| public | Yes | Yes | Yes |
| protected | Yes | Yes | No |
| private | Yes | No | No |

# Example – Rank Table Tennis Player



# 公有繼承：is-a關係

```
class RatedPlayer : public TableTennisPlayer
```

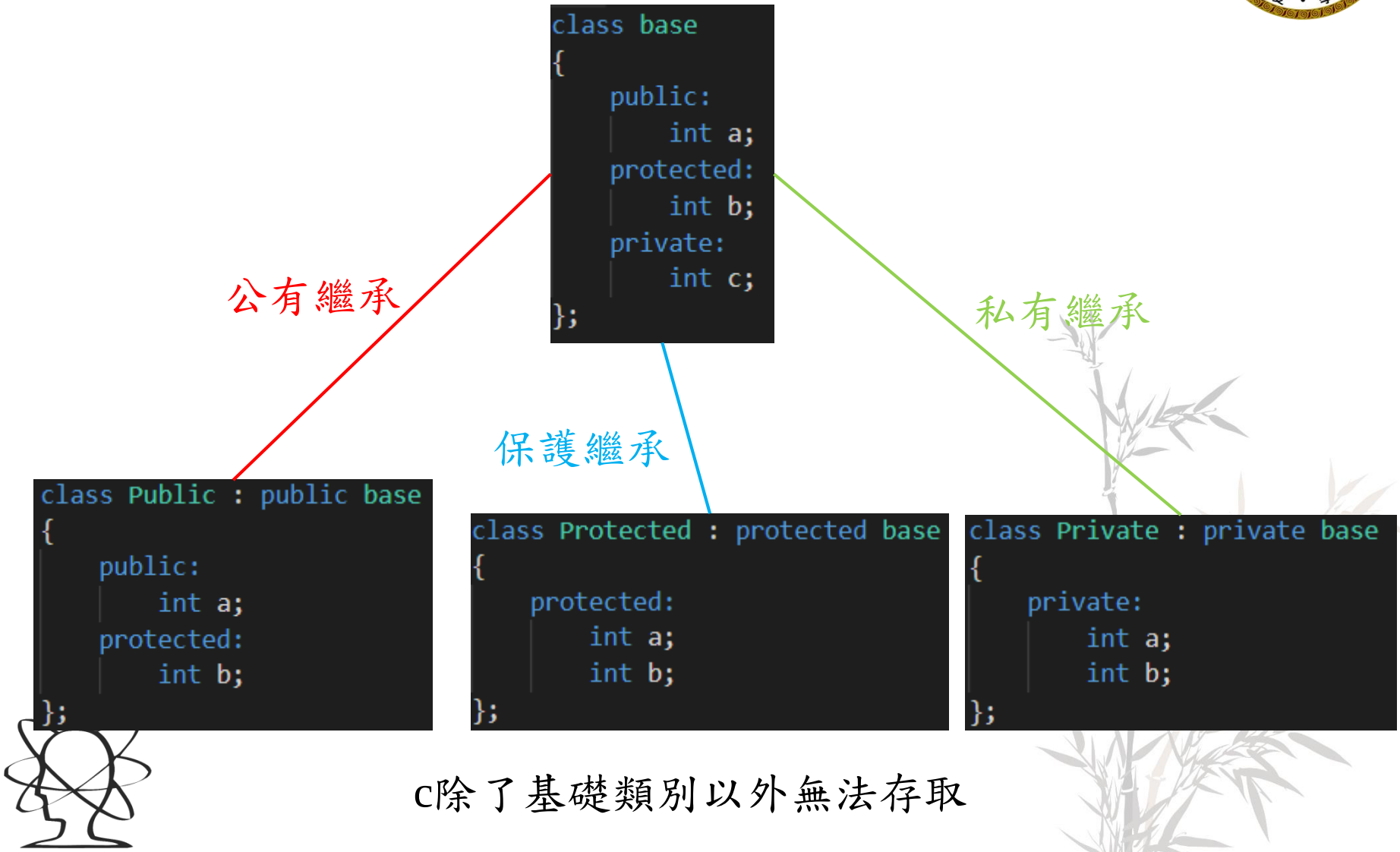



- 舉例：

香蕉是一種水果，

不代表水果就是香蕉

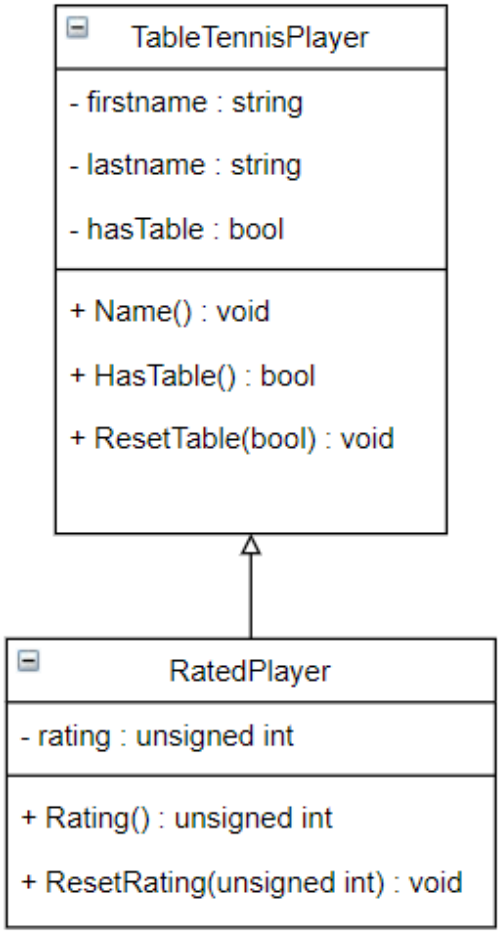
# 繼承




c除了基礎類別以外無法存取

# Pointer & Reference

- 基礎類別的指標與reference只能呼叫基礎類別的成員函數，無法呼叫衍生類別的成員函數
- 衍生類別的指標與reference可以呼叫基礎類別的成員函數



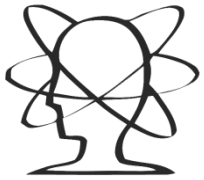
# 同名異式 Polymorphic

- 實作方法:
  - 在衍生類別中重新定義基礎類別的成員函數
  - 使用虛擬(Virtual)的成員函數

- 注意: Virtual只用於宣告，定義成員函數不用

```
virtual void ViewAcct() const;
```

```
void Brass::ViewAcct() const
{
    format initialState = setFormat();
    precis prec = cout.precision(2);

    cout << "Client: " << fullName << endl;
    cout << "Account Number: " << acctNum << endl;
    cout << "Balance: $" << balance << endl;

    restore(initialState, prec);
}
```



# Example – brass.h

## 基礎類別 Brass

- 成員變數
  - 客戶名稱 fullname
  - 帳號 acctNum
  - 目前餘額 balance
- 成員函數
  - 產生帳號
  - 存款
  - 提款
  - 顯示帳號資訊

## 衍生類別 BrassPlus

- 成員變數
  - 透支保護上限金額 maxLoan
  - 透支貸款利率 rate
  - 目前欠款 owesBank
- 成員函數
  - 提款(需有透支保護機制)
  - 顯示帳號資訊(含新增資訊)

# Example – brass.h

基礎類別：Brass

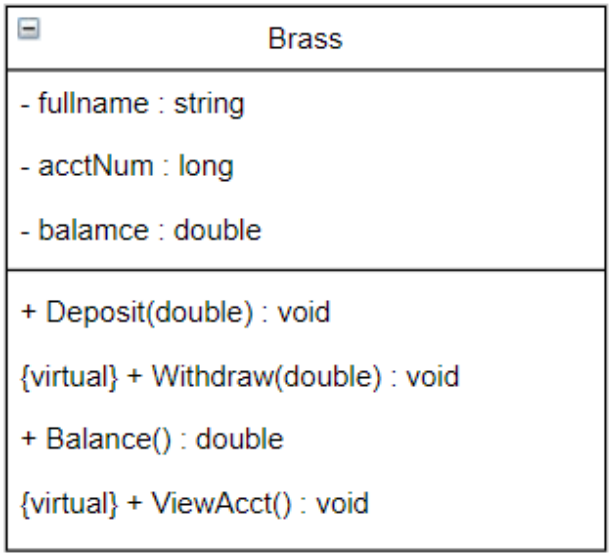
| Brass |
| --- |
| - fullname : string<br>- acctNum : long<br>- balamce : double |
| + Deposit(double) : void<br>{virtual} + Withdraw(double) : void<br>+ Balance() : double<br>{virtual} + ViewAcct() : void |

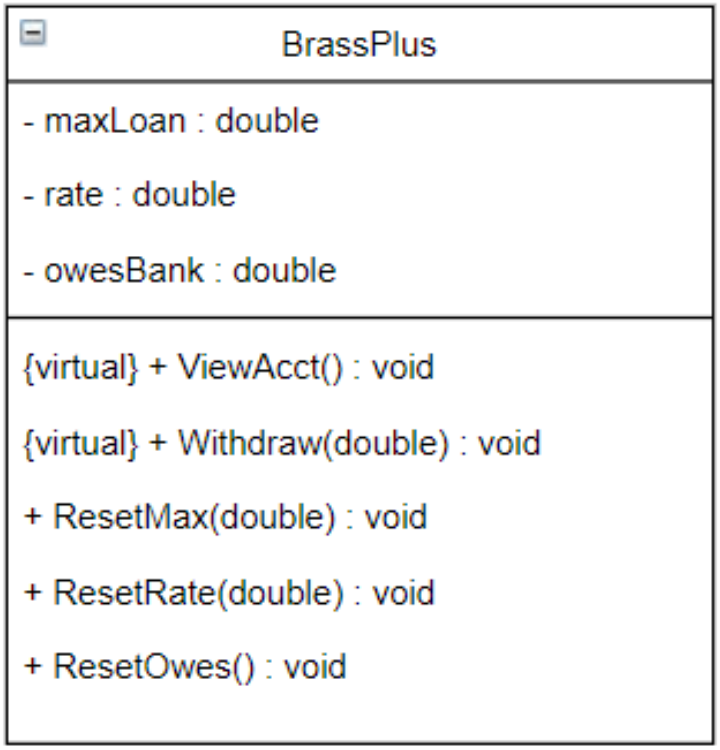
衍生類別：BrassPlus

| BrassPlus |
| --- |
| - maxLoan : double<br>- rate : double<br>- owesBank : double |
| {virtual} + ViewAcct() : void<br>{virtual} + Withdraw(double) : void<br>+ ResetMax(double) : void<br>+ ResetRate(double) : void<br>+ ResetOwes() : void |

# Example – brass.h重點

- 衍生類別加入三個新的私有成員與三個公用成員函數
- 基礎類別(Brass)與衍生類別(BrassPlus)都宣告ViewAcct()與Withdraw()成員函數並使用關鍵字virtual



# 若只使用靜態生成建構類別

- 有無virtual都沒差，程式會去尋找該類別的成員函數

```
Brass dom("Dominic Banker", 11224, 4183.45);
BrassPlus dot("Dorothy Banker", 12118, 2592.00);
dom.ViewAcct();       // use Brass::ViewAcct()
dot.ViewAcct();       // use BrassPlus::ViewAcct()
```

# Example – usebrass1.cpp

兩邊結果相同

## 無virtual

```
gitpod /workspace/Cplusplus-Primer-plus-writeup/Chapter13 $ ./test
Client: Porcelot Pigg
Account Number: 381299
Balance: $4000.00

Client: Horatio Hogg
Account Number: 382288
Balance: $3000.00
Maximum loan: $500.00
Owed to bank: $0.00
Loan Rate: 11.125%

Depositing $1000 into the Hogg Account:
New balance: $4000
Withdrawal amount of $4200.00 exceeds your balance.
Withdrawal canceled.
Piggy account balance: $4000
Withdrawing $4200 from the Hogg Account:
Bank advance: $200.00
Finance charge: $22.25
Client: Horatio Hogg
Account Number: 382288
Balance: $0.00
Maximum loan: $500.00
Owed to bank: $222.25
Loan Rate: 11.125%
```

## 有virtual

```
gitpod /workspace/Cplusplus-Primer-plus-writeup/Chapter13 $ ./usebrass1
Client: Porcelot Pigg
Account Number: 381299
Balance: $4000.00

Client: Horatio Hogg
Account Number: 382288
Balance: $3000.00
Maximum loan: $500.00
Owed to bank: $0.00
Loan Rate: 11.125%

Depositing $1000 into the Hogg Account:
New balance: $4000
Withdrawal amount of $4200.00 exceeds your balance.
Withdrawal canceled.
Piggy account balance: $4000
Withdrawing $4200 from the Hogg Account:
Bank advance: $200.00
Finance charge: $22.25
Client: Horatio Hogg
Account Number: 382288
Balance: $0.00
Maximum loan: $500.00
Owed to bank: $222.25
Loan Rate: 11.125%
```

# 若使用指標或reference呼叫類別

- 有無virtual有差

無

```
// behavior with non-virtual ViewAcct()
// method chosen according to reference type
Brass dom("Dominic Banker", 11224, 4183.45);
BrassPlus dot("Dorothy Banker", 12118, 2592.00);
Brass & b1_ref = dom;
Brass & b2_ref = dot;
b1_ref.ViewAcct();       // use Brass::ViewAcct()
b2_ref.ViewAcct();       // use Brass::ViewAcct()
```

有

```
// behavior with virtual ViewAcct()
// method chosen according to object type
Brass dom("Dominic Banker", 11224, 4183.45);
BrassPlus dot("Dorothy Banker", 12118, 2592.00);
Brass & b1_ref = dom;
Brass & b2_ref = dot;
b1_ref.ViewAcct();       // use Brass::ViewAcct()
b2_ref.ViewAcct();       // use BrassPlus::ViewAcct()
```

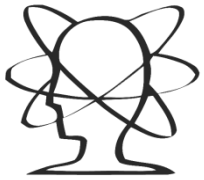
# Question

- 那為何不用BrassPlus型態的reference去呼叫dot?

```
// behavior with non-virtual ViewAcct()
// method chosen according to reference type
Brass dom("Dominic Banker", 11224, 4183.45);
BrassPlus dot("Dorothy Banker", 12118, 2592.00);
Brass & b1_ref = dom;
Brass & b2_ref = dot;      BrassPlus & b2_ref = dot;
b1_ref.ViewAcct();          // use Brass::ViewAcct()
b2_ref.ViewAcct();          // use Brass::ViewAcct()
                                Use BrassPlus:ViewAcct()
```

- Ans :

因為當資料量大時，不會逐一生成物件，而是使用陣列或動態生成(new)，但陣列與動態生成的特性是每個元素都需要是相同的資料型態



# Example – usebrass2.cpp

```
#include<iostream>
#include<string>
#include"brass.h"

const int CLIENTS = 4;

int main(void)
{
    Brass* p_clients[CLIENTS];
    string temp;
    long tempnum;
    double tempbal;
    char kind;

    for (int i = 0; i < CLIENTS; i++)
    {
        cout << "Enter client's name: ";
        getline(cin, temp);

        cout << "Enter client's account number: ";
        cin >> tempnum;

        cout << "Enter opening balance: $";
        cin >> tempbal;

        cout << "Enter 1 for Brass Amount or " << "2 for BrassPlus Account: ";
```

# 比較有無使用虛擬函式

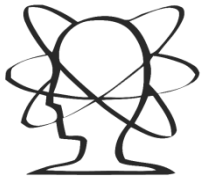
```
void showImf(Brass& s)
{
    s.ViewAcct();
    cout << endl;
}
```

ViewAcct無使用virtual
Brass b1(…);
BrassPlus b2(…);

b1、b2丟入showing
皆顯示Brass::ViewAcct()

ViewAcct有使用virtual
Brass b1(…);
BrassPlus b2(…);

b1、b2丟入showing
b1顯示Brass::ViewAcct()
b2顯示BrassPlus:: ViewAcct()



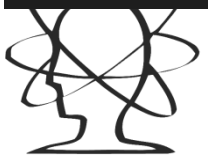
# ABC Design

- ABC (Abstract Base Class) 抽象基礎類別
- 舉例：

  假設今天要寫一個繪圖程式，此程式要能繪製圓與橢圓，圓是橢圓的特例，是一個長軸與短軸皆相同的橢圓，所以直覺上橢圓會是基礎類別，圓是橢圓的衍生類別

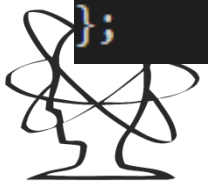
# Design - Ellipse

```
class Ellipse
{
    private:
        double x; // x-coordinate of the ellipse's center
        double y; // y-coordinate of the ellipse's center
        double a; // semimajor axis
        double b; // semiminor axis
        double angle; // orientation angle in degrees

    public:
        void Move(int nx, int ny) { x = nx; y = ny; }
        virtual double Area() const { return 3.14159 * a * b; }
        virtual void Rotate(double nang) { angle += nang; }
        virtual void Scale(double sa, double sb) { a *= sa; b *= sb; }
};
```



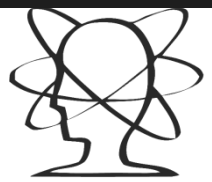
# Design – Circle : Ellipse

```
class Circle : public Ellipse
{
    private:
        double x; // x-coordinate of the ellipse's center
        double y; // y-coordinate of the ellipse's center
        double a; // semimajor axis
        double b; // semiminor axis
        double angle; // orientation angle in degrees

    public:
        void Move(int nx, int ny) { x = nx; y = ny; }
        virtual double Area() const { return 3.14159 * a * b; }
        virtual void Rotate(double nang) { angle = nang; }
        virtual void Scale(double sa, double sb) { a *= sa; b *= sb; }
};
```



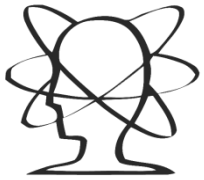
# Design - Circle

```
class Circle
{
    private:
        double x; // x-coordinate of the circle's center
        double y; // y-coordinate of the circle's center
        double r; // radius

    public:
        void Move(int nx, int ny) { x = nx; y = ny; }
        virtual double Area() const { return 3.14159 * r * r; }
        virtual void Scale(double sr) { r *= sr; }
};
```



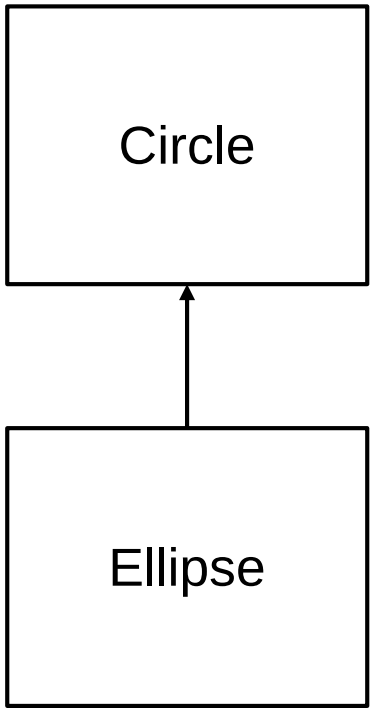
# Design – Ellipse : Circle

```
class Ellipse : public Circle
{
    private:
        double a; // semimajor axis
        double b; // semiminor axis
        double angle; // orientation angle in degrees

    public:
        virtual double Area() const { return 3.14159 * a * b; }
        void Rotate(double nang) { angle += nang; }
        virtual void Scale(double sa, double sb) { a *= sa; b *= sb; }
};
```

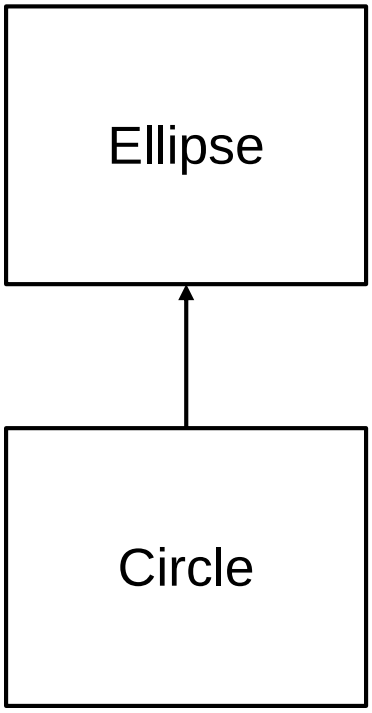


# How to Design?

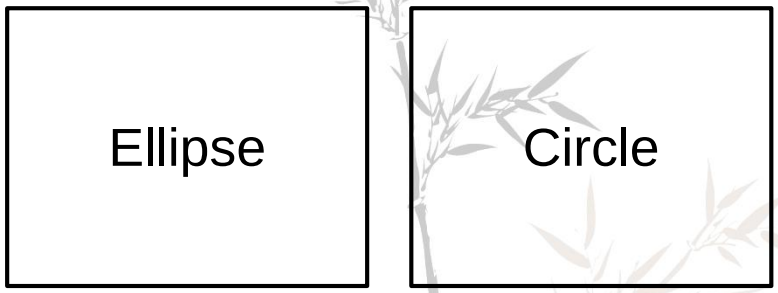
Method 1



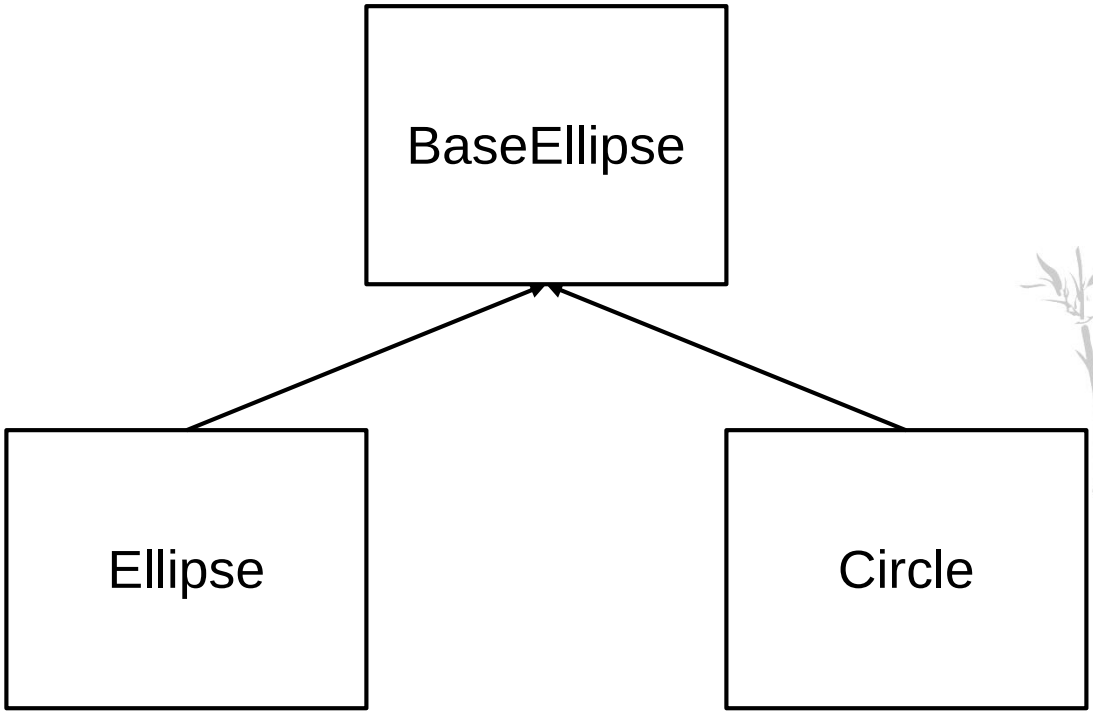

Method 2




Method 3



# ABC Design



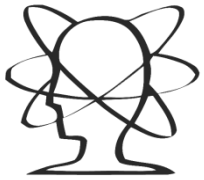
# Design - BaseEllipse

```
class BaseEllipse
{
    private:
        double x; // x-coordinate of center
        double y; // y-coordinate of center

    public:
        BaseEllipse(double x0 = 0, double y0 = 0) : x(x0),y(y0) {}
        virtual ~BaseEllipse() {}
        void Move(int nx, int ny) { x = nx; y = ny; }
        virtual double Area() const = 0; // a pure virtual function
        virtual void Scale() = 0;
};
```

純虛擬函式

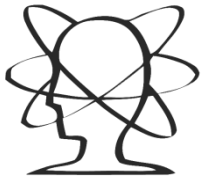
# Design – Circle : BaseEllipse

```
class Circle : public BaseEllipse
{
    private:
        double r; // radius

    public:
        virtual double Area() const { return 3.14159 * r * r; }
        virtual void Scale(double sr) { r *= sr; }
};
```

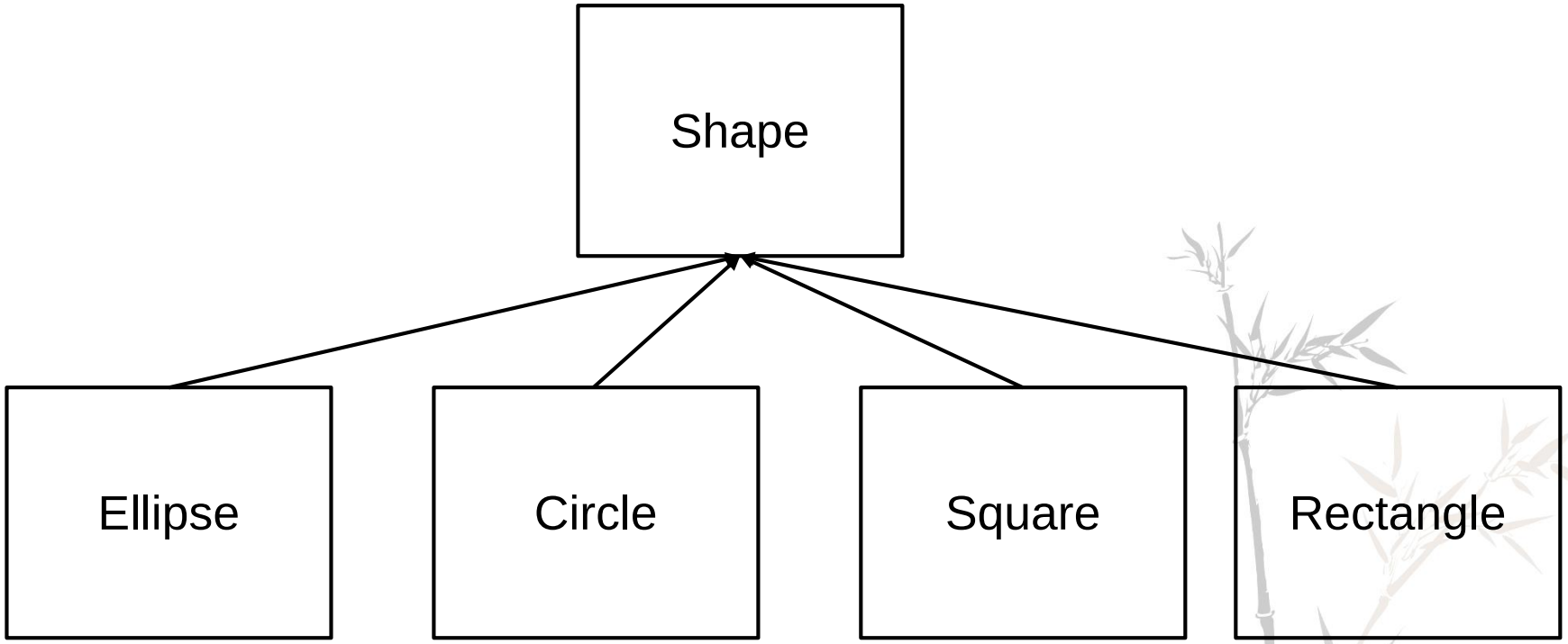


# Design – Ellipse : BaseEllipse

```
class Ellipse : public BaseEllipse
{
    private:
        double a; // semimajor axis
        double b; // semiminor axis
        double angle; // orientation angle in degrees
    public:
        virtual double Area() const { return 3.14159 * a * b; }
        void Rotate(double nang) { angle += nang; }
        virtual void Scale(double sa, double sb) { a *= sa; b *= sb; }
};
```



# ABC優點：擴充性佳



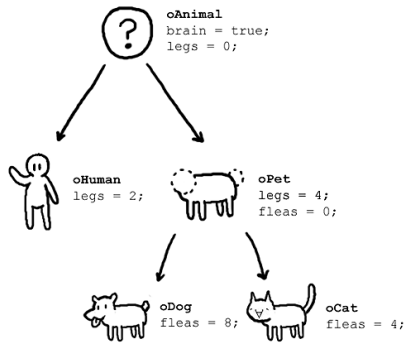
# MEME Time

Introduction to OOP

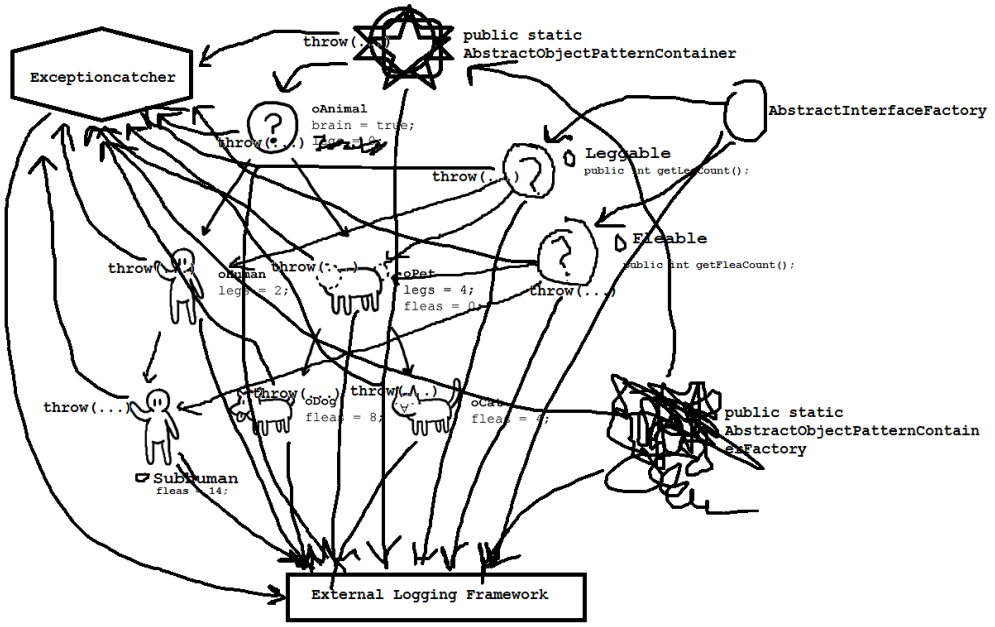
OOP in detail

# MEME Time

## What OOP users claim

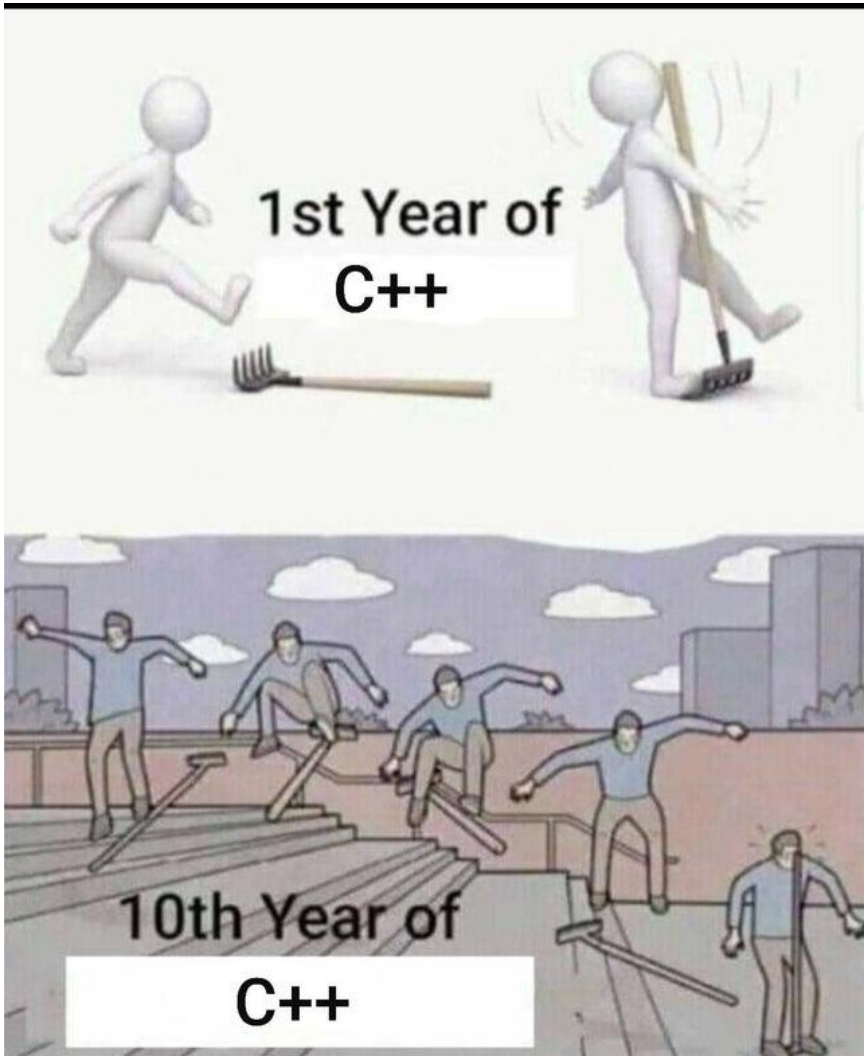



## What actually happens



# Q&A TIME

# MEME Time



# MEME Time

# Appendix A – Lab Environment

- Windows 10 – Visual Studio 2017
- Linux – Ubuntu 16.04、 Ubuntu 18.04
- Programming Language – Python ≧ 3.6, C++ 11

# Appendix B

- All code can be downloaded from Link
- https://github.com/Offliners/RobotLab-Cpp-Training-2022

# Thanks for your attention


NTU ME Robotics Lab.
臺大機械系機器人實驗室