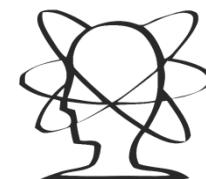




2023 C++ Training

報告人: 研一 王易騰



NTU ME Robotics Lab.
臺大機械系機器人實驗室



Outline



請你列出一堂兩小時C++程式語言課程的課程大綱。



當然！以下是一個可能的C++課程大綱，涵蓋了C++程式語言的基本知識，變數，資料型別，運算符號，控制結構和函數等基礎主題：

課程大綱

1. 課程介紹和基礎概念

- 課程目標和內容
- C++程式語言的歷史和發展
- 編譯器和開發環境介紹

2. 變數和資料型別

- 變數和常量的定義和使用
- 資料型別和宣告
- 整數、浮點數、布林值和字元型別
- 資料型別轉換

3. 運算符號

- 算數運算符號
- 關係運算符號
- 邏輯運算符號
- 指定運算符號

4. 控制結構

- 條件語句和if/else語句
- 選擇語句和switch語句
- 迴圈和while/do-while/for語句

5. 函數和指標

- 函數定義和調用
- 函數參數和回傳值
- 指標和指標運算符號
- 參考和引用

6. 陣列和字串

- 一維和多維陣列
- 字串定義和運算
- 字串函數和標準庫函數

7. 類和物件

- 類和物件的概念和定義
- 類的成員變數和成員函數
- 類的繼承和多型
- 標準庫和標準模板庫

8. 動態記憶體配置和異常處理

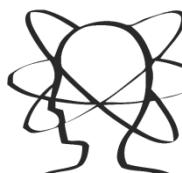
- new和delete運算符號
- 動態陣列和動態物件
- 異常處理和try/catch語句

9. 應用程式開發

- C++的應用程式開發
- 檔案輸入/輸出和流運算子
- C++程式語言的應用實例

10. 評估和結語

- 評估學生學習成果
- 未來發展和學習建議
- 課程結語和感謝詞





Outline

◆ C++ Basics

- | *Basic Structure*
- | *Data type & Array*
- | *Operator*
- | *Control Statement & Loop*
- | *Function*
- | *Namespace & Header*
- | *Argument*

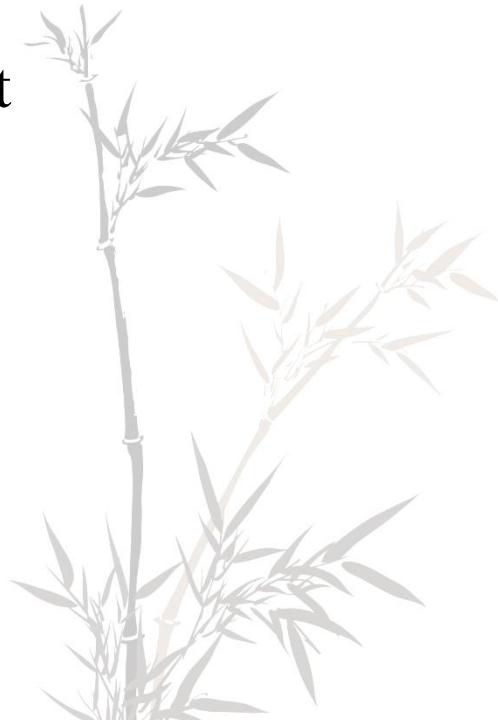


◆ Pointer & Reference

- | *Pointer Basics*
- | *Pointer & Variable/Array*
- | *Pass by Pointer*
- | *Reference*

◆ Class & Object

- | *Class Basics*
- | *Inheritance*
- | *Operator Overload*
- | *Polymorphism*
- | *Abstract Base Class*





Basic Structure

◆ 大家的第一個C++程式

HelloWorld.cpp

```
1 #include <iostream> ← 引入函式庫  
2  
3 int main() ← 主程式進入點  
4 {  
    命名空間  
    std::cout << "Hello World!";  
    return 0; ← 主程式結束點(0代表成功執行)  
7 }
```

```
~/2023-RobotLab-Cpp-Training  
└─ ./HelloWorld  
Hello World!
```



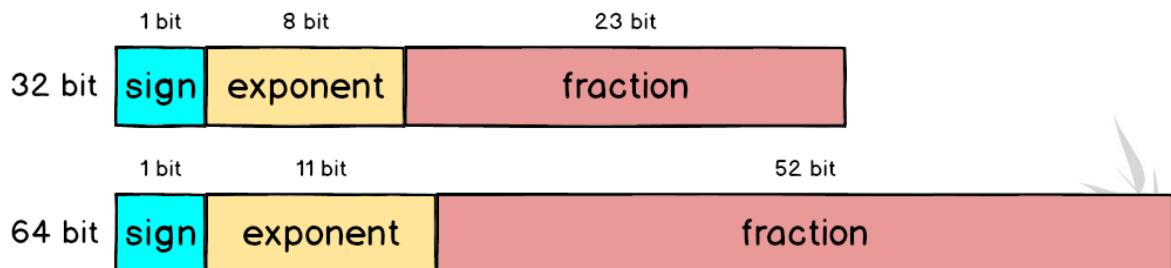


Data Type

◆ 兩種類型：整數、浮點數

類型	大小	範圍
bool	1 byte	0, 1/False, True
char	1 byte	-128~127
int	4 bytes	-2,147,483,648~2,147,483,647
long long	8 bytes	-9,223,372,036,854,775,808 ~9,223,372,036,854,775,807
float	4 bytes	1.17549e-38~3.40282e+38
double	8 bytes	2.22507e-308~1.79769e+308

※1 byte = 8 bits

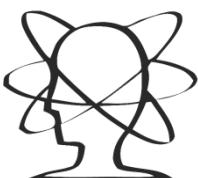




Data Type

- ◆ 不同型態所佔大小與系統位元相關，如16 bit架構中int為2 bytes，C語言規範了型態間大小關係。
- ◆ $\text{long} \geq \text{int} \geq \text{short}$

```
~/2023-RobotLab-Cpp-Training
./DataType
-----Size of Datatype-----
char: 1 byte.
bool: 1 byte.
int: 4 bytes.
long long: 8 bytes.
float: 4 bytes.
double: 8 bytes.
-----Range of Datatype-----
char:          -128 ~ 127
bool:          0 ~ 1
int:          -2147483648 ~ 2147483647
long long: -9223372036854775808 ~ 9223372036854775807
float:         1.17549e-38 ~ 3.40282e+38
double:        2.22507e-308 ~ 1.79769e+308
```





Data Type

- ◆ 使用 sizeof 運算子取得變數的位元大小、limits 裡 numeric_limits 的 min, max 函式取得型態值域範圍。

```
1 #include <iostream>
2 #include <iomanip>
3 #include <limits>
4
5 using namespace std;
6
7 int main()
8 {
9     cout << "-----Size of Datatype-----" << endl;
10    cout << setw(12) << "char: " << sizeof(char) << " byte." << endl;
11    cout << setw(12) << "bool: " << sizeof(bool) << " byte." << endl;
12    cout << setw(12) << "int: " << sizeof(int) << " bytes." << endl;
13    cout << setw(12) << "long long: " << sizeof(long long) << " bytes." << endl;
14    cout << setw(12) << "float: " << sizeof(float) << " bytes." << endl;
15    cout << setw(12) << "double: " << sizeof(double) << " bytes." << endl;
16
17    cout << "-----Range of Datatype-----" << endl;
18    cout << setw(12) << "char: " << setw(20) << int(numeric_limits<char>::min()) << " ~ " << int(numeric_limits<char>::max()) << endl;
19    cout << setw(12) << "bool: " << setw(20) << numeric_limits<bool>::min() << " ~ " << numeric_limits<bool>::max() << endl;
20    cout << setw(12) << "int: " << setw(20) << numeric_limits<int>::min() << " ~ " << numeric_limits<int>::max() << endl;
21    cout << setw(12) << "long long: " << setw(20) << numeric_limits<long long>::min() << " ~ " << numeric_limits<long long>::max() << endl;
22    cout << setw(12) << "float: " << setw(20) << numeric_limits<float>::min() << " ~ " << numeric_limits<float>::max() << endl;
23    cout << setw(12) << "double: " << setw(20) << numeric_limits<double>::min() << " ~ " << numeric_limits<double>::max() << endl;
24
25 }
```

DataType.cpp



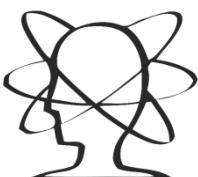
Basic IO



- ◆ 函式庫<iostream> (Input/Output Stream)
- ◆ 輸出: std::cout 搭配插入運算子 <<
- ◆ 輸入: std::cin 搭配擷取運算子 >>

BasicIO-1.cpp

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int input;
8
9     cout << "Input a integer number: ";
10    cin >> input;
11    cout << "Your input is: " << input;
12    return 0;
13 }
```





Basic IO

◆ 輸出結果

```
~/2023-RobotLab-Cpp-Training
./BasicIO-1
Input a integer number: 3
Your input is: 3
```

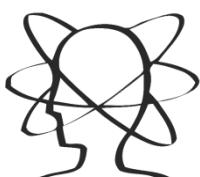
正常的輸出結果。

```
~/2023-RobotLab-Cpp-Training
./BasicIO-1
Input a integer number: 3.1415926
Your input is: 3
```

輸入float至int，小數部分被直接捨去。

```
~/2023-RobotLab-Cpp-Training
./BasicIO-1
Input a integer number: A
Your input is: 0
```

輸入char至int，造成cin失效。





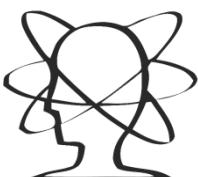
Basic IO

- ◆ 函式庫<iomanip> (IO Manipulation)
- ◆ 常用函式

```
1 setw(int) 指定輸入、出寬度  
2 setprecision(int) 指定輸入、出位數  
3 setbase(int) 指定輸入、出的進位制(只支援 8,10,16進制)  
4 setfill(char) 填滿輸入、出的空位，通常搭配setw()使用
```

- ◆ 格式旗標(Format Flag)

```
1 fixed 強制把浮點數表示成小數  
2 left 靠左對齊  
3 right 靠右對齊  
4 internal 置中對齊
```



Basic IO



```
~/2023-RobotLab-Cpp-Training
./BasicIO-2
-----Precision Control-----
Precision: 5
Without Fixed: 271.83
With     Fixed: 271.82818
-----Fill Emptyspace-----
|||||||||||||||||||||||||||Fill It!
```





Array

- ◆ 一次宣告N個變數，使用同個變數名稱並透過索引(index)來區別。

1 資料型態 變數名稱[長度]

- ◆ c方法未指定變數值，預設全初始化至0或空字元'\0'。

```
1 // Different array initialize method
2 int a[5];
3 int b[5] = {3, 45, 71, 13, 26};
4 int c[5] = {};
5 int d[5] = {1};
```

```
~/2023-RobotLab-Cpp-Training
./Array
-1010489792 32764 1070304616 21975 1
3 45 71 13 26
0 0 0 0 0
1 0 0 0 0
```

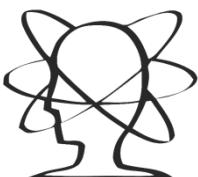
變數	b[0]*	b[1]	b[2]	b[3]	b[4]
數值	3	45	71	13	26

*陣列的索引由0開始



Operator

- ◆ 各種運算子有不同的優先級、結合性(方向性)
算數運算子: %, *, /, +, - (按優先及左至右排序)
關係運算子: >, <, >=, <=, ==, !=
邏輯運算子: &&, ||, !
賦值運算子: =, 各類變形
- ◆ 除賦值、單元運算子外大多順序都由左到右。\\





Operator

◆ 賦值運算子結合性為右到左

Operator-1.cpp

```
1 // Assign Operator
2 int i = 0, j = 0, k = 0;
3 k = 3.14;
4 // (i)
5 i = j = k; // assign j = k first, then i = j.
6 cout << "i = j = k" << endl;
7 cout << i << " " << j << " " << k << endl;
8 // (ii)
9 i = j = 0;
10 (i = j) = k; // assign i = j first, then i = k.
11 cout << "(i = j) = k" << endl;
12 cout << i << " " << j << " " << k << endl;
13 // (iii)
14 i = j = 0;
15 i = (j = k); // same as (i)
16 cout << "i = (j = k)" << endl;
17 cout << i << " " << j << " " << k << endl;
```

```
~/2023-RobotLab-Cpp-Training
./Operator-1
i = j = k
3 3 3
(i = j) = k
3 0 3
i = (j = k)
3 3 3
```





Operator

- ◆ 前置遞增運算子會先處理增量，而後置後處理增量，遞減亦然

Operator-2.cpp

```
1 int i = 0, j = 0;
2 // In(De)crement Operator
3 cout << "++i: " << ++i << " j++: " << j++ << endl;
4 cout << "i: " << i << " j: " << j << endl;
5 // Compound Assignment Operator
6 // a += b equal a = a + b
7 i *= 2; 複合賦值運算子
8 j += 3;
9 cout << "i: " << i << " j: " << j << endl;
10 i *= j;
11 cout << "i: " << i << " j: " << j << endl;
12 // Modulo Operator
13 cout << "i % 3 = " << i << " % 3 = " << i % 3 << endl;
```

```
~/2023-RobotLab-Cpp-Training
./Operator-2
++i: 1 j++: 0
i: 1 j: 1
i: 2 j: 4
i: 8 j: 4
i % 3 = 8 % 3 = 2
```





Operator

- ◆ 賦值、計算時的顯式、隱式轉換
- ◆ <typeinfo>的typeid().name()回傳資料類型

```
1 // Conversion
2 // Conversion when assigning value, implicit
3 char a; int b; float c;
4 a = b = c = 65.65;
5 cout << "a = b = c = 65.65" << endl;
6 cout << "c = float(65.65) => b = int(c)=65 => a = char(b) = A" << endl;
7 cout << "a: " << a << " b: " << b << " c: " << c << endl;
8
9 int i = 5, j = 3;
10 cout << "-----Direct Calculation-----" << endl;
11 cout << "i / j: " << i / j << "-" << typeid(i/j).name() << endl;
12 // Implicit Conversion
13 cout << "-----Implicit Conversion-----" << endl;
14 cout << "i / 3.0: " << i / 3.0 << "-" << typeid(i/3.0).name() << endl;
15 cout << "5.0 / j: " << 5.0 / j << "-" << typeid(5.0/j).name() << endl;
16 // Explicit Conversion
17 cout << "-----Explicit Conversion-----" << endl;
18 cout << "i / double(j): " << i / double(j) << "-" << typeid(i/double(j)).name() << endl;
19 cout << "double(i) / j: " << double(i) / j << "-" << typeid(double(i)/j).name() << endl;
```

Operator-3.cpp

```
~/2023-RobotLab-Cpp-Training
./Operator-3
a = b = c = 65.65
c = float(65.65) => b = int(c)=65 => a = char(b) = A
a: A b: 65 c: 65.65
-----Direct Calculation-----
i / j: 1-i
-----Implicit Conversion-----
i / 3.0: 1.66667-d
5.0 / j: 1.66667-d
-----Explicit Conversion-----
i / double(j): 1.66667-d
double(i) / j: 1.66667-d
```





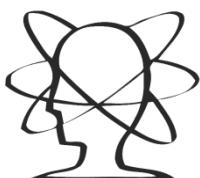
Operator

◆ 關係、邏輯運算子

```
1 // Relational Operator
2 // >, <, >=, <=, ==, !=
3 int i, j;
4 i = 7; j = 4;
5 // Return True:1 False:0
6 cout << "i > j: " << (i > j) << endl;
7 cout << "i == j: " << (j == j) << endl;
8 cout << "j < 4: " << (j < 4) << endl;
9
10 // Logical Operator
11 // &&, ||, !
12 cout << "i > 5 && j < 5: " << (i > 5 && j < 5) << endl;
13 cout << "i > 5 || j > 5: " << (i > 5 || j > 5) << endl;
14 cout << !(i > 6): " << !(i > 6) << endl;
```

Operator-4.cpp

```
~/2023-RobotLab-Cpp-Training
./Operator-4
i > j: 1
i == j: 1
j < 4: 0
i > 5 && j < 5: 1
i > 5 || j > 5: 1
!(i > 6): 0
```





If-else Statement

- ◆ 如果...就...(否則...)
- ◆ 如果...就...否則如果...就...(否則...)

e.g. 如果市場有蛋就買十顆蛋；否則如果超市有蛋就買一盒蛋；否則就不買。

```
1 if(condition)
2 {
3     statement;
4 }
5 else
6 {
7     statement;
8 }
```

```
1 if(condition)
2 {
3     statement;
4 }
5 else if(condition)
6 {
7     statement;
8 }
9 else
10 {
11     statement;
12 }
```



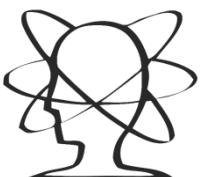


If-else Statement

- ◆ 使用結構控制的語法建議使用大括號{}，將statement包起來。

```
1  if(x > 5)
2      if(y > 5)
3          cout << "x, y > 5\n";
4  else
5      cout << "x <= 5\n";
6 // Which is right?
7 if(x > 5)
8     if(y > 5)
9         cout << "x, y > 5\n";
10 else
11     cout << "x <= 5\n";
```

```
1  if(x > 5)
2  {
3      if(y > 5)
4          cout << "x, y > 5\n";
5  }
6 else
7 {
8     cout << "x <= 5\n";
9 }
```





If-else Statement

```
1 int StudentScore;
2
3 cout << "Your Score: ";
4 cin >> StudentScore;
5
6 if(StudentScore >= 60){
7     cout << "You Passed!" << endl;
8 }
9 else{
10     cout << "You Failed, see you next semester ; C" << endl;
11 }
12
13 if(StudentScore >= 80){
14     cout << "You got an A." << endl;
15 }
16 else if(StudentScore < 80 && StudentScore >= 70){
17     cout << "You got a B." << endl;
18 }
19 else if(StudentScore < 70 && StudentScore >= 60){
20     cout << "You got a C." << endl;
21 }
```

Ifelse.cpp

```
~/2023-RobotLab-Cpp-Training
./Ifelse
Your Score: 74
You Passed!
You got a B
```





Switch Statement

- ◆ 根據整數運算式(expression)的值，執行不同的陳述句。
- ◆ case 1:類似 if (condition==1)，default: 類似else處理不符合其他條件的情況。

```
1  switch(condition)
2  {
3      case 1:
4          statement;
5          break;
6      case 2:
7          statement;
8          break;
9      default:
10         statement;
11         break;
12 }
```





Switch Statement

- ◆ 適合處理簡單、重複判斷的條件，若要選擇特定範圍也可用a ... b表示。

```
1 int StudentScore;           SwitchCase-1.cpp
2
3 cout << "Your Score: ";
4 cin >> StudentScore;
5
6 switch(StudentScore / 10)
7 {
8     case 10: 無break往下個case的
9     case 9:   statement執行
10    case 8:   cout << "You got an A." << endl;
11        break;
12    case 7:
13        cout << "You got a B." << endl;
14        break;
15    case 6:
16        cout << "You got a C." << endl;
17        break;
18    default:
19        cout << "You Failed." << endl;
20        break;
21 }
22 }
```



```
1 int StudentScore;           SwitchCase-2.cpp
2
3 cout << "Your Score: ";
4 cin >> StudentScore;
5
6 switch(StudentScore)
7 {
8     case 80 ... 100:
9         cout << "You got an A." << endl;
10        break;
11    case 70 ... 79:
12        cout << "You got a B." << endl;
13        break;
14    case 60 ... 69:
15        cout << "You got a C." << endl;
16        break;
17    default:
18        cout << "You Failed." << endl;
19        break;
20 }
```



Switch Statement

◆ 輸出結果

SwitchCase-1.cpp

```
~/2023-RobotLab-Cpp-Training
./SwitchCase-1
Your Score: 90
You got an A.

~/2023-RobotLab-Cpp-Training
./SwitchCase-1
Your Score: 74
You got a B.

~/2023-RobotLab-Cpp-Training
./SwitchCase-1
Your Score: 65
You got a C.

~/2023-RobotLab-Cpp-Training
./SwitchCase-1
Your Score: 54
You Failed.
```

SwitchCase-2.cpp

```
~/2023-RobotLab-Cpp-Training
./SwitchCase-2
Your Score: 90
You got an A.

~/2023-RobotLab-Cpp-Training
./SwitchCase-2
Your Score: 74
You got a B.

~/2023-RobotLab-Cpp-Training
./SwitchCase-2
Your Score: 65
You got a C.

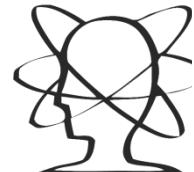
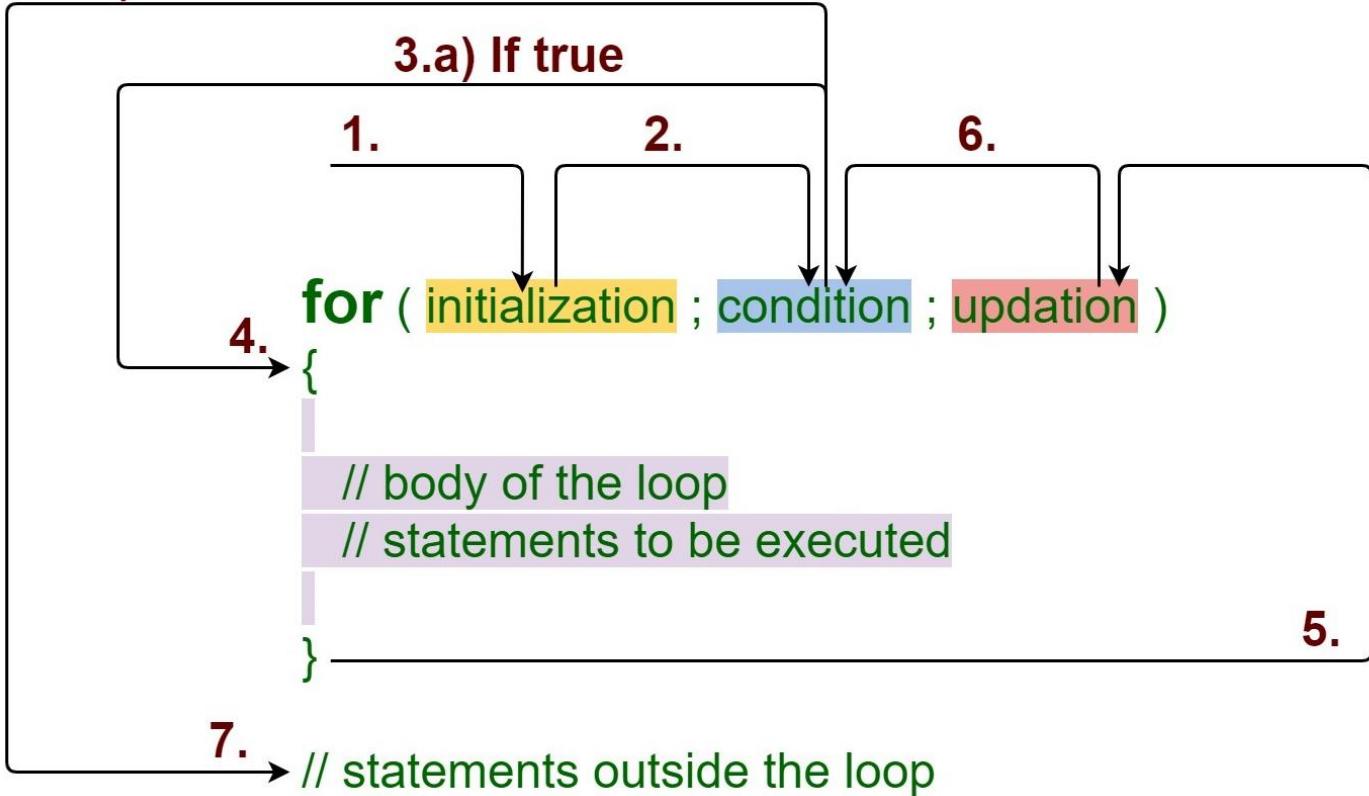
~/2023-RobotLab-Cpp-Training
./SwitchCase-2
Your Score: 54
You Failed.
```





For Loop

3.b) If false



Loop



- ◆ for loop
- ◆ 簡單的遍尋x，並印出數值。

```
1 int x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
2  
3 cout << "x: ";  
4 // int: 4bytes, x: 4*10 bytes, x[0]: 4 bytes  
5 for (int i = 0; i < sizeof(x)/sizeof(x[0]); i++){  
6     cout << x[i] << " ";  
7 }  
8 cout << "\n";
```

```
~/2023-RobotLab-Cpp-Training  
./ForLoop  
x: 1 2 3 4 5 6 7 8 9 10
```



Loop



- ◆ 常用遞增/減來進行迴圈控制，但可以是任何運算式，也可以用,分隔執行多個運算式。

ForLoop.cpp

```
1 int x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
2  
3 cout << "x: "  
4 // int: 4bytes, x: 4*10 bytes, x[0]: 4 bytes  
5 for (int i = 0; i < sizeof(x)/sizeof(x[0]); x[i] = 0, i++){  
6     cout << x[i] << " "  
7 }  
8 cout << "\n";  
9 // print again  
10 cout << "x: "  
11 for (int i = 0; i < sizeof(x)/sizeof(x[0]); i++){  
12     cout << x[i] << " "  
13 }  
14 cout << "\n";
```

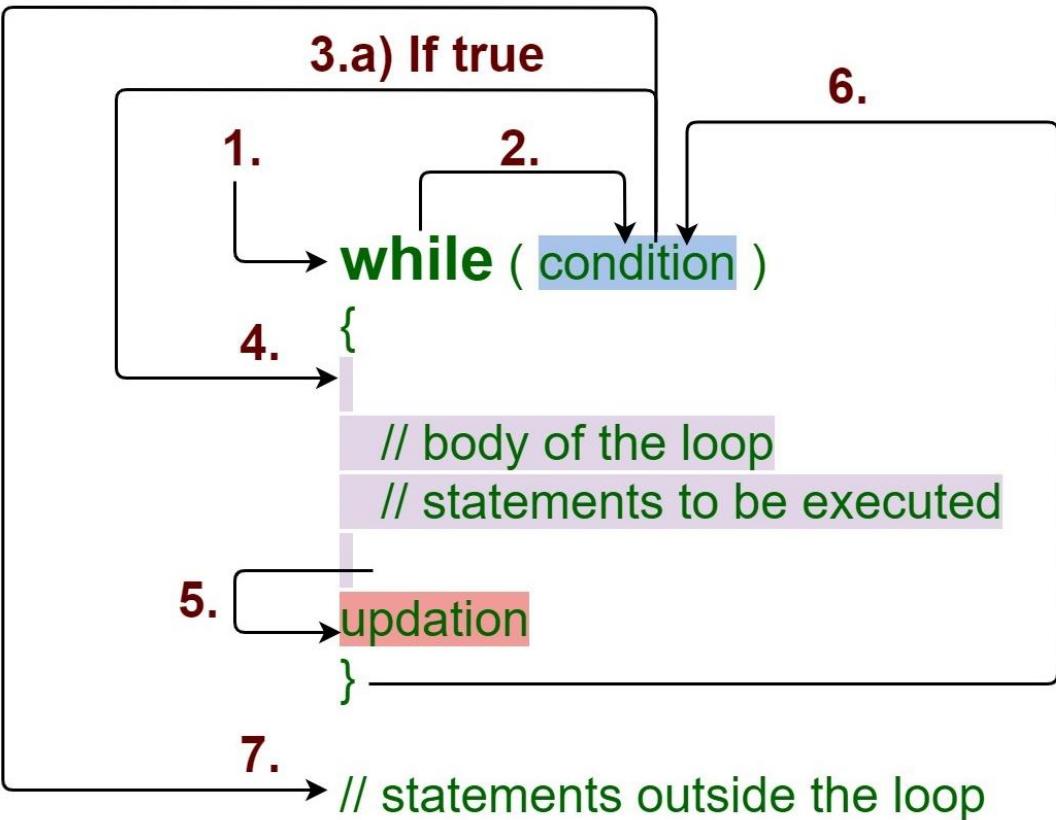
```
~/2023-RobotLab-Cpp-Training  
./ForLoop  
x: 1 2 3 4 5 6 7 8 9 10  
x: 0 0 0 0 0 0 0 0 0 0
```





While Loop

3.b) If false



Loop



- ◆ while不包含變數初始化，必須要宣告過才能拿去當作條件。

WhileLoop.cpp

```
1 int i = 0, x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
2  
3 cout << "x: ";  
4 // cannot initialize here  
5 while(i < 10){  
6     cout << x[i] << " ";  
7     i++;  
8 }  
9 cout << "\n";
```

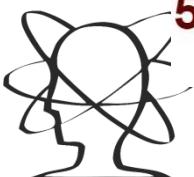
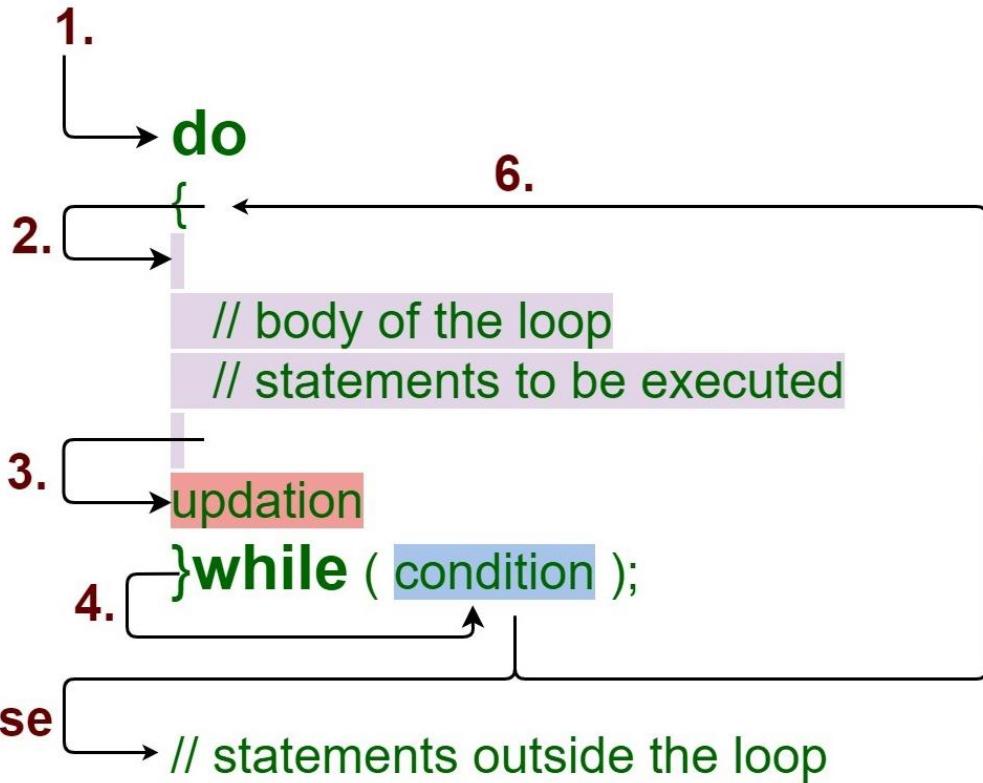
```
~/2023-RobotLab-Cpp-Training  
./WhileLoop  
x: 1 2 3 4 5 6 7 8 9 10
```



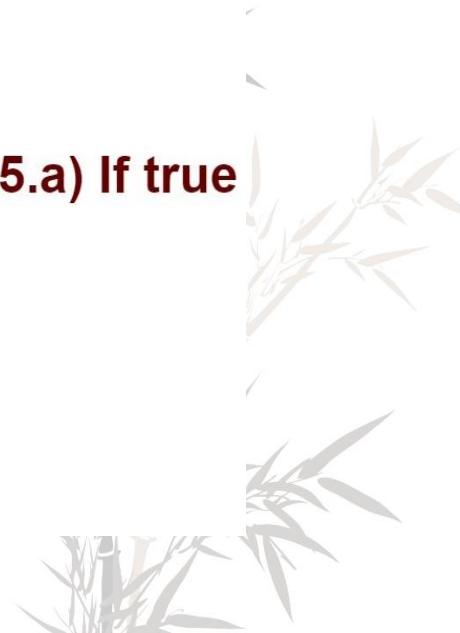


Loop

Do - While Loop



5.b) If false



Loop



◆ do-while迴圈無論條件都會先執行一次

```
DoWhile.cpp
1 int i = 0, x[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
2 // while Loop
3 cout << "x: ";
4 while(i > 0 && i < 10){
5     cout << x[i] << " ";
6     i++;
7 }
8 cout << "\n";
9 // do while Loop
10 i = 0; // reset i
11 cout << "x: ";
12 do{
13     cout << x[i] << " ";
14     i++;
15 }while(i > 0 && i < 10);
16 cout << "\n";
```

```
~/2023-RobotLab-Cpp-Training
  ./DoWhile
x:
x: 1 2 3 4 5 6 7 8 9 10
```





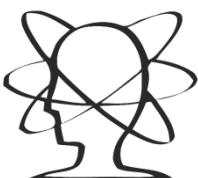
Function

- ◆ 將主程式中一段程式碼包裝為函數，可以自訂輸入/出的類型
- ◆ 包含回傳值、函式名稱、參數列、函式主體

```
#include<iostream>
```

```
void displayNum(int n1, double n2) {  
    // code  
}  
  
int main() {  
    ...  
    displayNum(num1, num2);  
    ...  
}
```

function
call





Function

◆ 直接在main()前實作函式

```
1 #include <iostream>
2
3 using namespace std;
4
5 // using Loop
6 long long factorial(int n){
7     long long fact = 1;
8     for (int i = n; i > 0; i--){
9         fact *= i;
10    }
11    return fact;
12 }
13
14 int main()
15 {
16     int N;
17     cout << "Input N to calculate N!: ";
18     cin >> N;
19     cout << N << "!" = " << factorial(N) << endl;
20     return 0;
21 }
```

Function-1.cpp

```
~/2023-RobotLab-Cpp-Training
./Function-1
Input N to calculate N!: 5
5! = 120
```





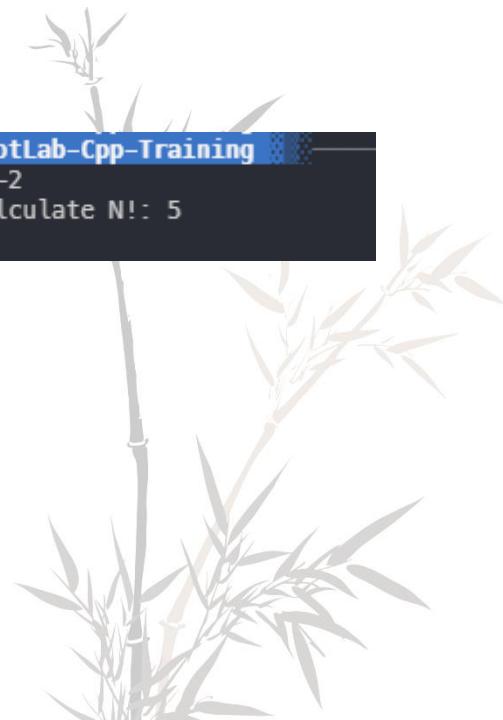
Function

◆ 我們也可以選擇先宣告後再進行實作

```
1 #include <iostream>
2
3 using namespace std;
4
5 long long factorial(int); 函式宣告
6
7 int main()
8 {
9     int N;
10    cout << "Input N to calculate N!: ";
11    cin >> N;
12    cout << N << "!" = " << factorial(N) << endl;
13    return 0;
14 }
15 // recursive method
16 long long factorial(int n){ 函式實作
17     if (n > 0){
18         return n * factorial(n-1);
19     }
20     else if (n == 0){return 1;}
21 }
```

Function-2.cpp

```
~/2023-RobotLab-Cpp-Training
./Function-2
Input N to calculate N!: 5
5! = 120
```



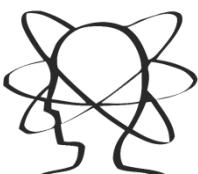


Function

◆ 如果沒宣告直接在main()後實作？

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int N;
8     cout << "Input N to calculate N!: ";
9     cin >> N;
10    cout << N << "!" = " << factorial(N) << endl;
11    return 0;
12 }
13 // using Loop
14 long long factorial(int n){
15     long long fact = 1;
16     for (int i = n; i > 0; i--){
17         fact *= i;
18     }
19     return fact;
20 }
```

```
Starting build...
/usr/bin/g++ -fdiagnostics-color=always -g /home/relu/2023-RobotLab-Cpp-Training/Function-1.cpp -o /home/relu/2023-RobotLab-Cpp-Training/Function-1
/home/relu/2023-RobotLab-Cpp-Training/Function-1.cpp: In function 'int main()':
/home/relu/2023-RobotLab-Cpp-Training/Function-1.cpp:12:28: error: 'factorial' was not declared in this scope
cout << N << "!" = " << factorial(N) << endl;
^~~~~~
Build finished with error(s).
```





Namespace

- ◆ 避免同名變數、函式的情況，搭配範圍解析運算子`::`表示屬於哪個命名空間

```
1 #include <iostream>          NameSpace.cpp
2
3 using namespace std;
4
5 namespace A {
6     int x = 2022;
7     void dog(void){
8         cout << "Woof!" << endl;
9     }
10 }
11
12 namespace B {
13     int x = 2023;
14     void dog(void){
15         cout << "Let me do it for you." << endl;
16     }
17 }
18
19 int main()
20 {
21     cout << A::x << " " << B::x << endl;
22     A::dog();
23     B::dog();
24     return 0;
25 }
```

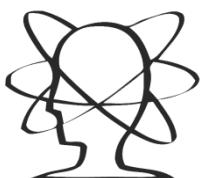
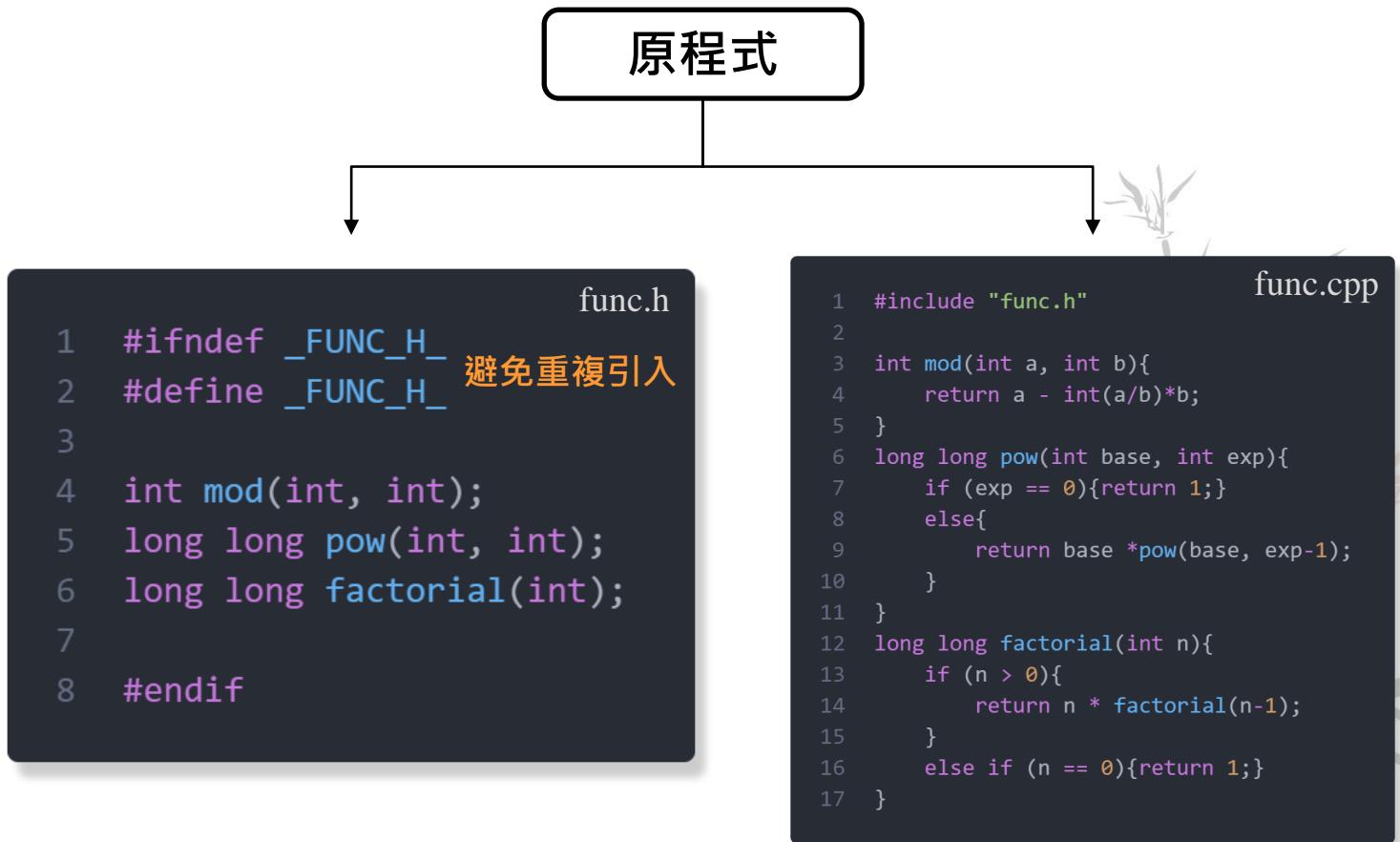
```
~/2023-RobotLab-Cpp-Training
./NameSpace
2022 2023
Woof!
Let me do it for you.
```





Header

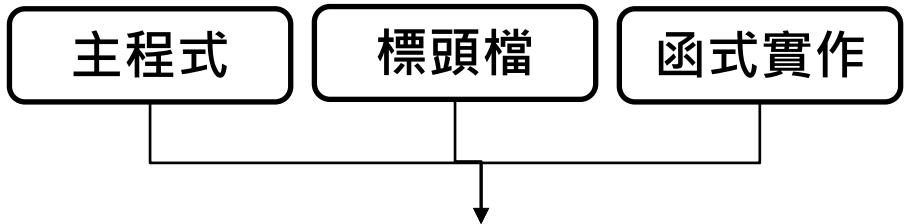
- ◆ 將函式宣告(.h)與實作(.cpp)分別儲存，需要使用再引入到程式中





Header

- ◆ 重新組合後，即使不重新定義函式也能直接呼叫



```
1 #include <iostream>
2 #include "func.h"
3 #include "func.cpp"
4
5 using namespace std;
6
7 int main()
8 {
9     cout << "5 % 3 = " << mod(5, 3) << endl;
10    cout << "2^6 = " << pow(2, 6) << endl;
11    cout << "6! = " << factorial(6) << endl;
12    return 0;
13 }
```

Header.cpp

```
~/2023-RobotLab-Cpp-Training
./Header
5 % 3 = 2
2^6 = 64
6! = 720
```





Argument

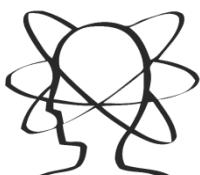
- ◆ 有些main括弧中不是空白，而有類似這樣的變數

```
1 int main(int argc, char *argv[])
      char **argv
```

```
Administrator: 命令提示字元
C:\Windows\system32>ping

使用方式: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
           [-r count] [-s count] [[-j host-list] | [-k host-list]]
           [-w timeout] [-R] [-S srcaddr] [-c compartment] [-p]
           [-4] [-6] target_name

選項:
-t          Ping 指定的主機，直到停止。
           若要查看統計資料並繼續，請按 Control-Break;
           若要停止，請按 Control-C。
-a          將位址解析為主機名稱。
-n count    要傳送的回應要求數目。
-l size     傳送緩衝區大小。
-f          在封包中設定 Don't Fragment 旗標 (僅 IPv4)。
-i TTL      存留時間。
-v TOS      服務類型 (僅 IPv4)。此設定已過時，而且對 IP 標頭中的
           服務類型欄位沒有影響)。
-r count    記錄路由以供計算躍點 (僅 IPv4)。
-s count    供計算躍點的時間戳記 (僅 IPv4)。
-j host-list 雜散的主機清單的來源路由 (僅 IPv4)。
-k host-list 嚴格的主機清單來源路由 (僅 IPv4)。
-w timeout   每個回覆的等候逾時 (單位為毫秒)。
-R          也使用路由標頭測試反向路由 (僅 IPv6)。
           根據 RFC 5095，已不再使用此路由標頭。如果使用此標頭，某些
           系統可能會捨棄回應要求。
-S srcaddr  要使用的來源位址。
-c compartment 路由區間識別碼。
-p          對 Hyper-V 網路虛擬化提供者位址執行 Ping。
-4          強制使用 IPv4。
```





Argument

- ◆ 命令列引數: argc紀錄引數數量, argv紀錄引數字串

```
1 #include <iostream>
2
3 using namespace std;
4
5 int GCD(int, int);
6
7 int main(int argc, char *argv[])
8 {
9     if (argc != 3){
10         cout << "Please use command " << argv[0] << " [int1] [int2]" << endl;
11         exit(1);
12     }
13     int a = atoi(argv[1]);
14     int b = atoi(argv[2]);
15     cout << "gcd(" << a << ", " << b << ") = " << GCD(a, b) << endl;
16     return 0;
17 }
18
19 int GCD(int a, int b){
20     if (a % b == 0)
21         return b;
22     else
23         return GCD(b, a % b);
24 }
```

Argument.cpp

```
~/2023-RobotLab-Cpp-Training
./Argument 141 9
gcd(141, 9) = 3
```

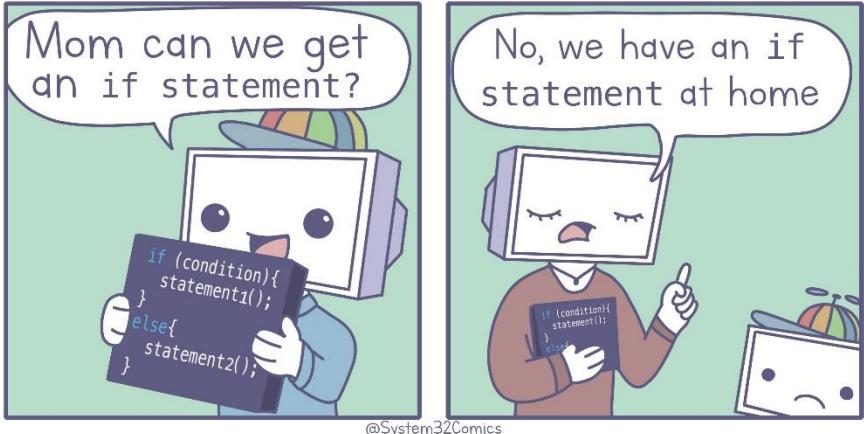


argc	argv[0]	argv[1]	argv[2]
3	Argument	141	9

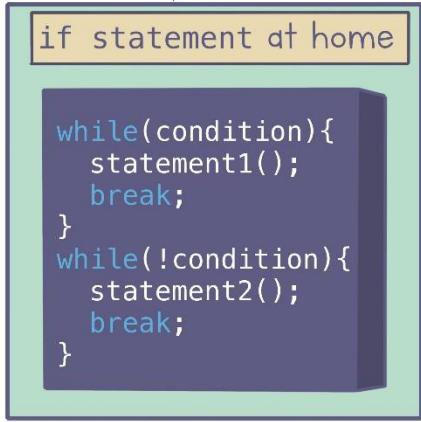




Break Time!



@System32Comics



System32Comics





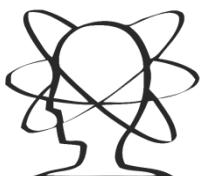
Pointer Basics

- ◆ 指針：儲存變數(記憶體)位置的變數
- ◆ 取址運算子 &：回傳變數的位置
- ◆ 提取運算子 *：回傳該位置的變數值

Pointer-1.cpp

```
1 int x = 4;
2
3 cout << "x's address: " << &x;
4 cout << ", x's value: " << x << endl;
```

```
1 // pointer declaration: type *name/ type* name
2 int *x_ptr;
3 // address-of operator &
4 x_ptr = &x;
5 cout << "x_ptr point to: " << x_ptr;
6 // dereference operator *
7 cout << ", it's value is: " << *x_ptr << endl;
```



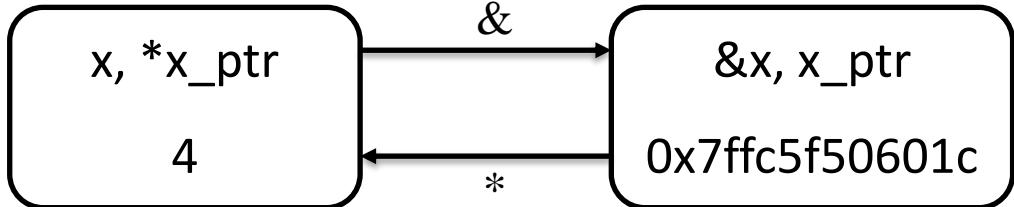


Pointer Basics

◆ 輸出結果

```
~/2023-RobotLab-Cpp-Training  
./Pointer-1  
x's address: 0x7ffc5f50601c, x's value: 4  
x_ptr point to: 0x7ffc5f50601c, it's value is: 4
```

◆ 指標與所指變數關係

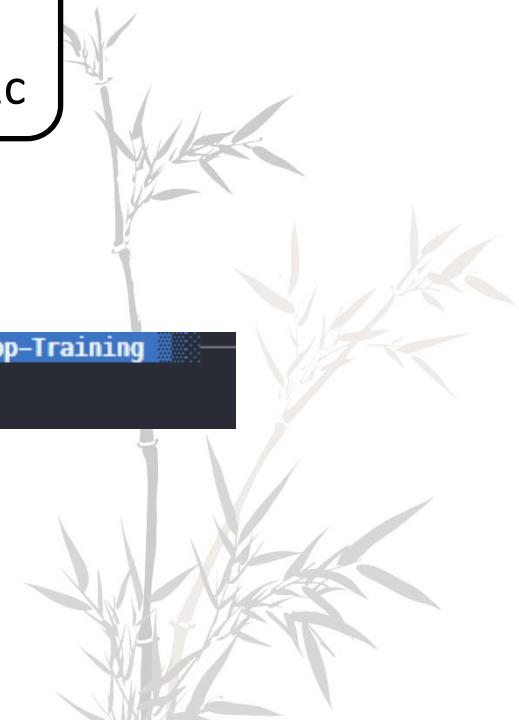
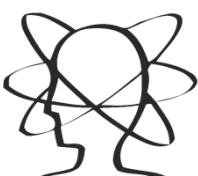


◆ 如果不賦予指標位置？

```
1 int *ptr;  
2 cout << "ptr's address: " << ptr << endl;
```

```
~/2023-RobotLab-Cpp-Training  
./Pointer-test  
ptr's address: 0
```

被編譯器分配至空指標(沒有指向任何東西)
，調用 *ptr 會導致程式 crash。





Pointer Basics

◆ 透過指標間接更改變數值

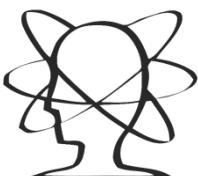
Pointer-2.cpp

```
1 int x = 4;
2 int *p = &x;
3
4 cout << "x: " << x << " *p: " << *p << endl;
5 *p = 5; // edit x's value by pointer
6 cout << "x: " << x << " *p: " << *p << endl;
```

```
~/2023-RobotLab-Cpp-Training
./Pointer-2
x: 4 *p: 4
x: 5 *p: 5
```

◆ 和直接更改x有什麼不同？

沒有(;́•ω•̀)。



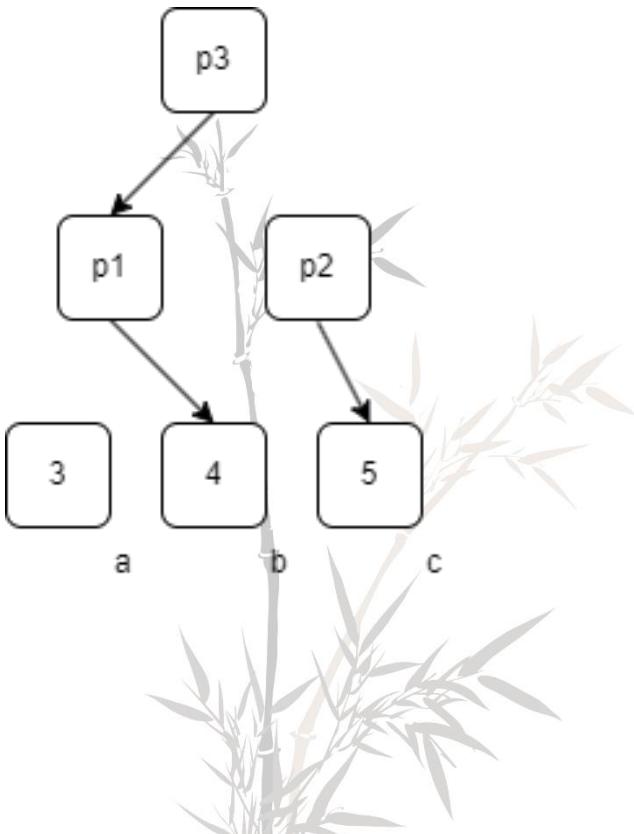


Pointer and Variable

◆ 小測驗

Pointer-3.cpp

```
1 int *p1, *p2;
2 int **p3;
3 int a = 3, b = 4, c = 5;
4 p1 = &b;
5 p2 = &c;
6 p3 = &p1;
7
8 // quiz time!
9 ++*p1;
10 p1 = &a;
11 **p3 = 10;
12 *p2 += 3;
13
14 cout << a << " " << b << " " << c << endl;
```



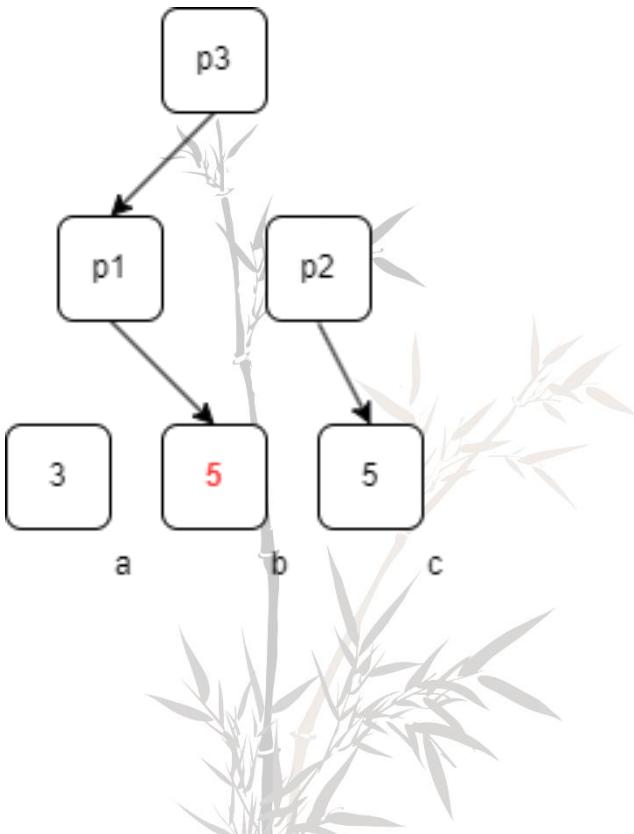


Pointer and Variable

◆ 小測驗

Pointer-3.cpp

```
1 int *p1, *p2;
2 int **p3;
3 int a = 3, b = 4, c = 5;
4 p1 = &b;
5 p2 = &c;
6 p3 = &p1;
7
8 // quiz time!
9 ++*p1;
10 p1 = &a;
11 **p3 = 10;
12 *p2 += 3;
13
14 cout << a << " " << b << " " << c << endl;
```



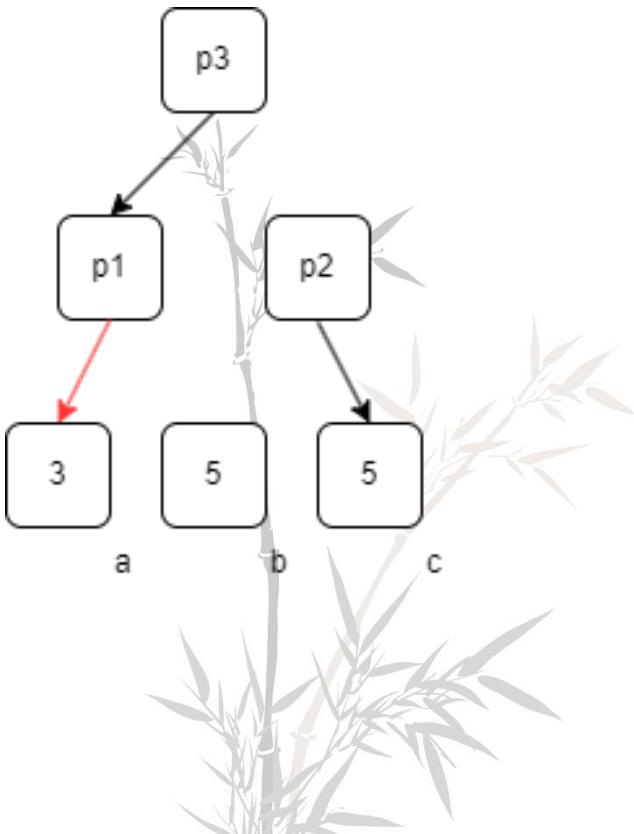


Pointer and Variable

◆ 小測驗

Pointer-3.cpp

```
1 int *p1, *p2;
2 int **p3;
3 int a = 3, b = 4, c = 5;
4 p1 = &b;
5 p2 = &c;
6 p3 = &p1;
7
8 // quiz time!
9 ++*p1;
10 p1 = &a;
11 **p3 = 10;
12 *p2 += 3;
13
14 cout << a << " " << b << " " << c << endl;
```



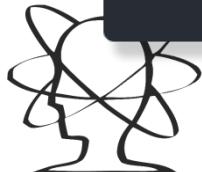
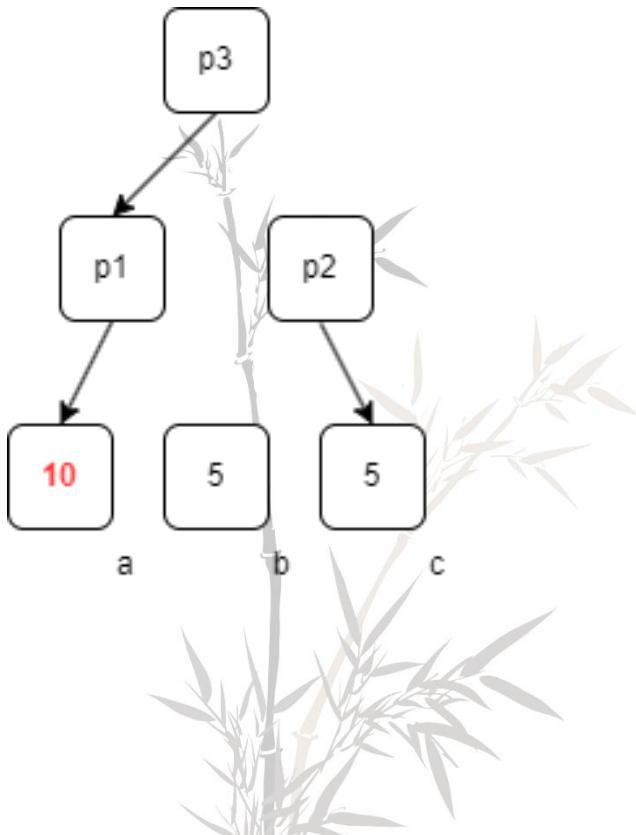


Pointer and Variable

◆ 小測驗

Pointer-3.cpp

```
1 int *p1, *p2;
2 int **p3;
3 int a = 3, b = 4, c = 5;
4 p1 = &b;
5 p2 = &c;
6 p3 = &p1;
7
8 // quiz time!
9 ++*p1;
10 p1 = &a;
11 **p3 = 10;
12 *p2 += 3;
13
14 cout << a << " " << b << " " << c << endl;
```



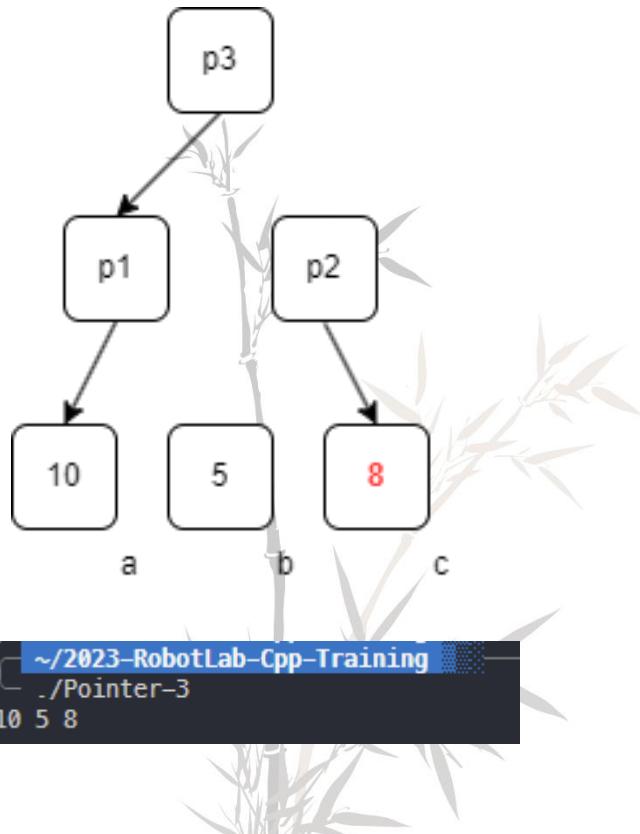


Pointer and Variable

◆ 小測驗

```
1 int *p1, *p2;
2 int **p3;
3 int a = 3, b = 4, c = 5;
4 p1 = &b;
5 p2 = &c;
6 p3 = &p1;
7
8 // quiz time!
9 ++*p1;
10 p1 = &a;
11 **p3 = 10;
12 *p2 += 3;
13
14 cout << a << " " << b << " " << c << endl;
```

Pointer-3.cpp



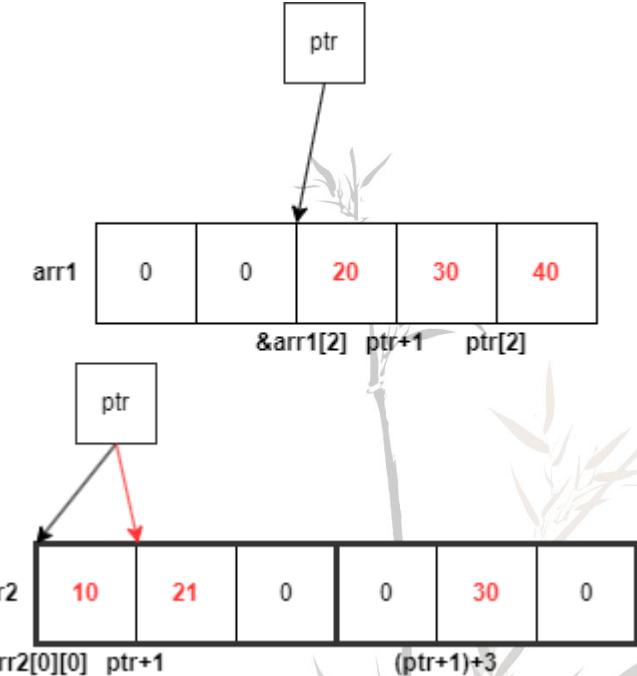


Pointer and Array

- ◆ 由於陣列記憶體配置為連續，除了透過直接指定index外，也可以透過指針來取得陣列數值

Pointer-4.cpp

```
1 int arr1[5] = {}, arr2[2][3] = {};
2 int *ptr = nullptr;
3
4 ptr = &arr1[2];
5 // ptr+1 = &arr1[2+1]
6 // ptr[2] = arr1[2+2]
7 *ptr = 20;
8 *(ptr+1) = 30;
9 ptr[2] = 40;
10
11 ptr = &arr2[0][0];
12 *ptr = 10;
13 // *++ptr = *(ptr + 1)
14 *++ptr = 20;
15 // ++*ptr = ++(*ptr)
16 ++*ptr;
17 ptr[3] = 30;
```



```
~/2023-RobotLab-Cpp-Training
./Pointer-4
arr1: 0 0 20 30 40
arr2: 10 21 0 | 0 0 30 |
```



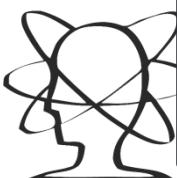
Pass by Pointer(address value)

- ◆ 為什麼要用指標？ A: 函式無法直接修改變數值...etc

Pointer-5.cpp

```
1 #include <iostream>
2
3 using namespace std;
4
5 void swap(int a, int b){
6     int temp = a;
7     a = b;
8     b = temp;
9 }
10
11 void swap_ptr(int *a, int *b){
12     int temp = *a;
13     *a = *b;
14     *b = temp;
15 }
16
17
18 int main()
19 {
20     int i = 4, j = 6;
21     swap(i, j);
22     cout << "Pass by value" << endl;
23     cout << "i: " << i << " j: " << j << endl;
24
25     i = 4, j = 6; // reset i, j
26     swap_ptr(&i, &j);
27     cout << "Pass by pointer" << endl;
28     cout << "i: " << i << " j: " << j << endl;
29     return 0;
30 }
```

```
~/2023-RobotLab-Cpp-Training
./Pointer-5
Pass by value
i: 4 j: 6
Pass by pointer
i: 6 j: 4
```





Reference

- ◆ 參考(reference)也能間接改變變數值，但必須在宣告時同時初始化，且不能重新初始化

```
1 #include <iostream>
2
3 using namespace std;
4
5 void swap_ref(int &a, int &b){
6     int temp = a;
7     a = b;
8     b = temp;
9 }
10
11 int main()
12 {
13     int a = 10, b = 7;
14     int *p = &a;
15     int &ref = a; // ref is also a itself
16     cout << "a: " << a << " *p: " << *p << " ref: " << ref << endl;
17
18     *p = 12;
19     cout << "a: " << a << " *p: " << *p << " ref: " << ref << endl;
20
21     ref++;
22     cout << "a: " << a << " *p: " << *p << " ref: " << ref << endl;
23
24     // pass by reference
25     cout << "a: " << a << " b: " << b << endl;
26     swap_ref(a, b);
27     cout << "a: " << a << " b: " << b << endl;
28     return 0;
29 }
```

Reference.cpp

```
~/2023-RobotLab-Cpp-Training
./Reference
a: 10 *p: 10 ref: 10
a: 12 *p: 12 ref: 12
a: 13 *p: 13 ref: 13
a: 13 b: 7
a: 7 b: 13
```



Class Basics

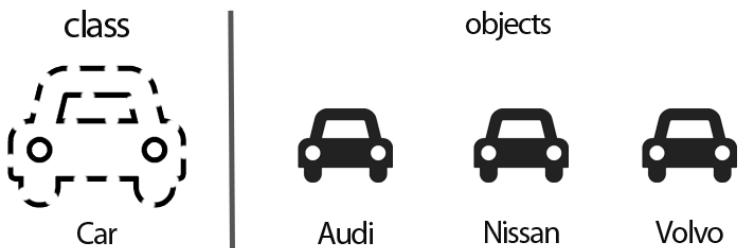
◆ 什麼是物件(Object)？

日常生活中的物品，並且有對應的屬性、行為
eg. 碗盤、狗、汽車...

1. 屬性: 碗盤大小、狗的毛色、汽車馬力
2. 行為: 碗盤盛食物、狗吠、汽車移動

◆ 什麼是類別(Class)？

C++ 中將程式包裝成物件的框架





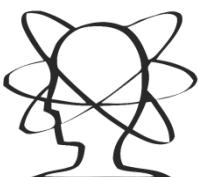
Class Basics

◆ 類別宣告方式

```
1 class ClassName{  
2     private: ←  
3     // Class members...  
4     public:  
5     // Class members...  
6 };
```

類別名稱
訪問修飾符
類別成員

- ◆ 類別成員：實現物件的屬性、行為
 - 1. 資料成員(Data member)
 - 2. 成員函式(Member function)





Class Basics

◆ 訪問修飾符: 決定存取成員能如何被存取

- ◆ private: 禁止類別外部存取
- ◆ public: 允許類別外部存取
- ◆ protect: 禁止類別外部存取，但允許繼承的類別存取

◆ 預設的訪問修飾符為private

```
1 class ClassName{  
2     int x;  
3     int y;  
4     void print_xy(){  
5         cout << "x: " << x << " y: " << y << endl;  
6     }  
7 };  
8  
9 int main()  
10 {  
11     ClassName a;    成員存取運算子。  
12     a.x = 10;        (指標用->)  
13     a.y = 5;  
14     a.print_xy();  
15     return 0;  
16 }
```

```
/home/relu/2023-RobotLab-Cpp-Training/Class.cpp:19:7: error: 'int ClassName::x' is private within this context  
      a.x = 10;  
           ^  
/home/relu/2023-RobotLab-Cpp-Training/Class.cpp:5:9: note: declared private here  
      int x;  
           ^  
/home/relu/2023-RobotLab-Cpp-Training/Class.cpp:20:7: error: 'int ClassName::y' is private within this context  
      a.y = 5;  
           ^  
/home/relu/2023-RobotLab-Cpp-Training/Class.cpp:6:9: note: declared private here  
      int y;  
           ^  
/home/relu/2023-RobotLab-Cpp-Training/Class.cpp:21:16: error: 'void ClassName::print_xy()' is private within this context  
      a.print_xy();  
           ^  
/home/relu/2023-RobotLab-Cpp-Training/Class.cpp:7:10: note: declared private here  
      void print_xy(){  
           ^~~~~~
```



Class Basics

◆ 類別建構函式(Class constructor)

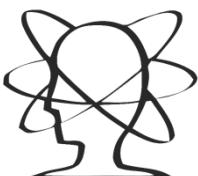
類別產生時，會呼叫此函式來做類別初始化，**此函式名稱需與類別同名稱**，由於無回傳值，所以不需要宣告型態void

◆ 類別解構函式(Class destructor)

類別在消失的時候，會自動呼叫解構函式來清除物件，**此函式名稱需為~類別名稱**，僅會自動清除靜態生成的成員，透過new動態生成的成員則需要delete清除

```
1 class Person{
2     private:
3         string name;
4         int age;
5
6     public:
7         Person(){
8             cout << "A person class is created!\n";
9         }
10        ~Person(){
11            cout << "A person class is cleared.\n";
12        }
13    };
```

讓在程式呼叫建/解構
函式時印出對應輸出





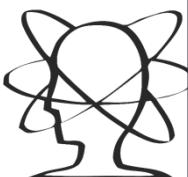
Class Basics

- ◆ 我們在這基礎上把person的資料成員宣告為私有(private)成員，進行封裝(encapsulation)讓使用者透過公有函式成員進行資料存取。

```
1 class Person{
2     private:
3         string name;
4         int age;
5     public:
6         Person(){
7             cout << "A person class is created!\n";
8         }
9         ~Person(){
10            cout << "A person class is cleared.\n";
11        }
12         void set_age(int a){age = a;}
13         void set_name(string n){name = n;}
14         void print_info(){
15             cout << "Name: " << name << " age: " << age << endl;
16         }
17     };
18
19 int main()
20 {
21     Person p1;
22     string input;
23     cout << "Please input your name: ";
24     cin >> input;
25     p1.set_name(input);
26
27     cout << "Please input your age: ";
28     cin >> input;
29     p1.set_age(stoi(input));
30
31     p1.print_info();
32     return 0;
33 }
```

Class-1.cpp

```
~/2023-RobotLab-Cpp-Training
./Class-1
A person class is created!
Please input your name: relu
Please input your age: 22
Name: relu age: 22
A person class is cleared.
```





Class Basics

- ◆ 成員初始化列表(Member initialization list)
- ◆ 透過建構函式參數賦值
- ◆ 結合以上兩者

```
1 // initialization list
2 Person(): name("John"), age(18){} 記得結尾加上大括號
3 // assign with constructor input
4 Person(string Name = "John", int Age = 18){
5     name = Name;
6     age = Age;
7 }
8 // Combine both
9 Person(string Name = "John", int Age = 18): name(Name), age(Age){}
```

```
~/2023-RobotLab-Cpp-Training
./Class-1
Name: John age: 18
```





UML 統一塑模語言

◆ 統一塑模語言（Unified Modeling Language，縮寫UML）是非專利的第三代塑模和規約語言。UML是一種開放的方法，用於說明、視覺化、構建和編寫一個正在開發的、物件導向的、軟體密集系統的製品的開放方法。

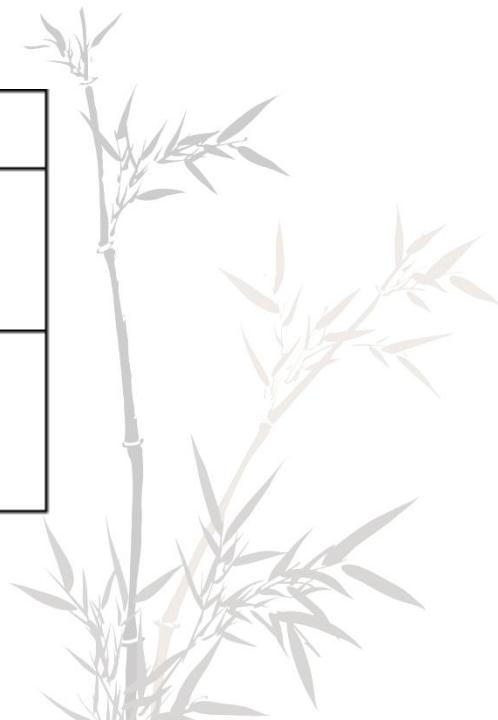
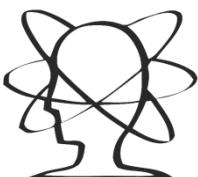
◆ 類別圖(Class Diagram)

+ : 表示公有成員

- : 表示私有成員

: 表示保護成員

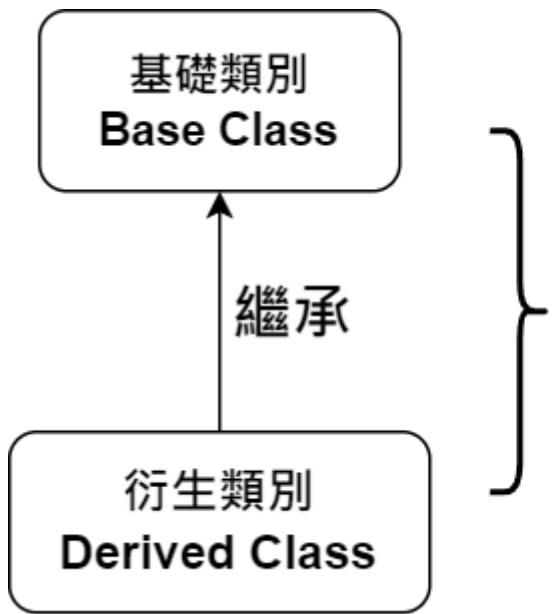
Classname
+ public var : type - private var : type # protected var : type
+ Function1(args type) : type - Function2(args type) : type # Function3(args type) : type



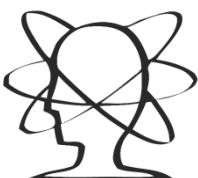


Inheritance

- ◆ 從現有類別繼承成員，衍生出新類別



1. 重複利用基礎類別
2. 新增/修改屬性、動作





Inheritance

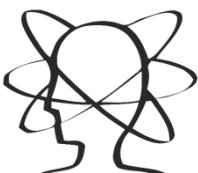
◆ 從Person類別衍生出Student類別

```
1 class Person{
2     private:
3         string name;
4         int age;
5     public:
6         Person(string Name = "John", int Age = 18): name(Name), age(Age){}
7         ~Person(){}
8         void set_age(int a){age = a;}
9         int get_age(){return age;}
10        void set_name(string n){name = n;}
11        string get_name(){return name;}
12        void print_info(){
13             cout << "Name: " << name << " age: " << age << endl;
14        }
15    };
```

Class-2.cpp

```
1 class Student: public Person{ Class 衍生類別: 繼承種類 基本類別
2     private:
3         string department;
4     public:
5         Student(string D = "unknown", string Name = "John", int Age = 18):
6             department(D), Person(Name, Age){}
7         Student(string D , const Person& P): department(D), Person(P){}
8         void set_department(string D){department = D;}
9         void print_student_info(){
10             print_info();
11             cout << "Major: " << department << ".\n";
12         }
13     };
```

由Person的參考
建構Student





Inheritance

◆ 繼承種類權限

```
1 class Base{  
2     public:  
3         int a;  
4     protected:  
5         int b;  
6     private:  
7         int c;  
8 }
```

衍生類別無法存取
基礎類別的private成員

```
1 class Public  
2 : public Base{  
3     public:  
4         int a;  
5     protected:  
6         int b;  
7 }
```

與Base相同

```
1 class Protected  
2 : protected Base{  
3     protected:  
4         int a;  
5         int b;  
6 }
```

皆為保護成員

```
1 class Private  
2 : private Base{  
3     private:  
4         int a;  
5         int b;  
6 }
```

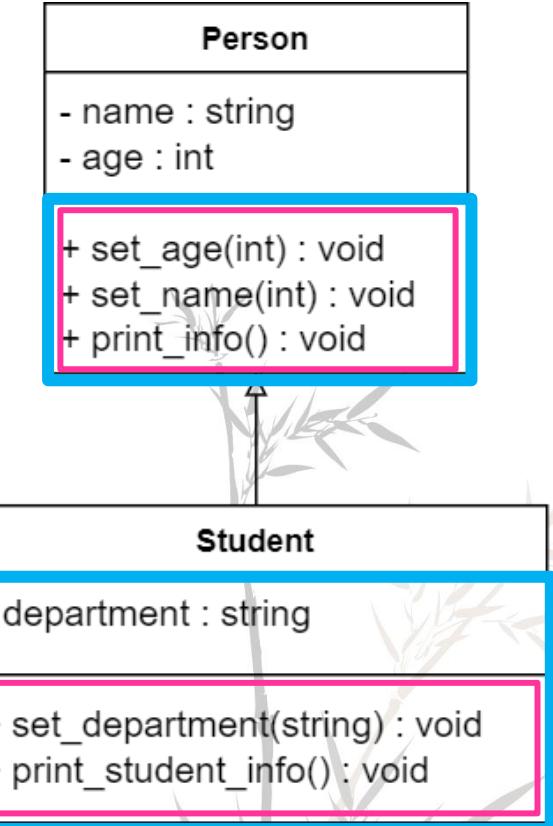
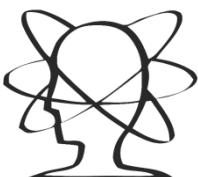
皆為私人成員



Inheritance

- ◆ Student透過公有繼承Person，因此藍框中的成員可以被Student存取(name, age無法直接被Student存取)
- ◆ 紅框中則是對於使用者能直接存取的成員
- ◆ 使Student成員直接存取Person的私人成員會造成錯誤

```
1 void print_student_info(){  
2     // print_info();  
3     cout << "Name: " << name << " age: " << age << endl;  
4     cout << "Major: " << department << ".\n";  
5 }
```





Inheritance

◆ 兩種宣告衍生類別的方式

```
Class-2.cpp
1 Student s1("Mechanical Engineering", "relu", 22);
2 s1.print_student_info();
3
4 Person p;
5 Student s2("Electrical Engineering", p);
6 s2.print_student_info();
```

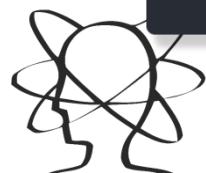
```
~/2023-RobotLab-Cpp-Training
./Class-2
Name: relu age: 22
Major: Mechanical Engineering.
Name: John age: 18
Major: Electrical Engineering.
```

◆ 衍生類別透過基礎類別的setFunction更新基礎類別的private資料

```
1 void Student::set_info(string D, string Name, int Age){
2     department = D;
3     this->set_age(Age);    this指標指到s2
4     this->set_name(Name); 再呼叫Person的函式
5 }
```

```
1 Person p;
2 Student s2("Electrical Engineering", p);
3 s2.set_info("Computer Science", "Jack", 21);
4 s2.print_student_info();
```

```
~/2023-RobotLab-Cpp-Training
./Class-2
Name: relu age: 22
Major: Mechanical Engineering.
Name: Jack age: 21
Major: Computer Science.
```

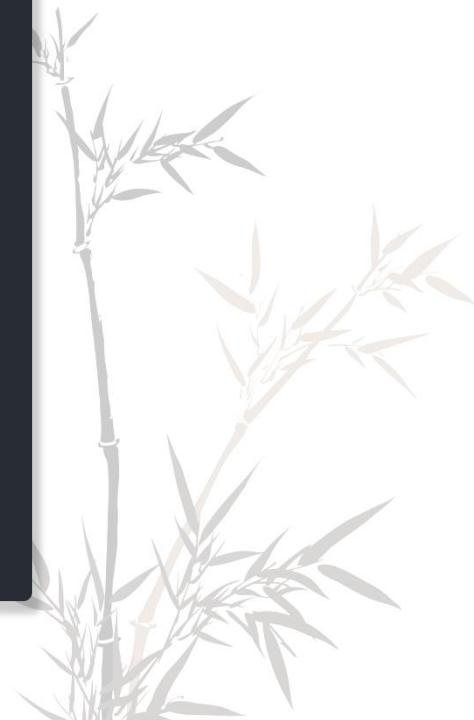
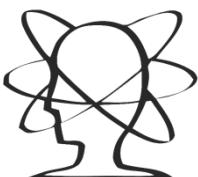




Operator Overload

- ◆ 我們已經會利用類別來建立物件了，但如果我們如果像以下直接對物件使用運算子操作會如何？

```
1 class Vector {  
2     private:  
3         int x,y;  
4     public:  
5         Vector(): x(0), y(0){};  
6         Vector(int X,int Y): x(X), y(Y){}  
7         void print(){cout << x << "," << y << endl;}  
8     };  
9  
10    int main(){  
11        Vector a (5,2);  
12        Vector b (3,1);  
13        Vector c;  
14        c = a + b;  
15        c.print();  
16        return 0;  
17    }
```





Operator Overload

- ◆ 我們可以針對運算子進行重載，定義運算子對於該類別的操作

回傳型態 運算子 輸入參數

```
1 Vector operator+(Vector param){  
2     Vector temp; // (0,0)  
3     temp.x = x + param.x;  
4     temp.y = y + param.y;  
5     return (temp);  
6 }
```

重載實作





Operator Overload

- ❖ 幾乎所有運算元都可以進行重載，例如直接讓物件透過插入運算子(<<)讓cout印出物件資訊

```
1 class Vector {                                OpOverload.cpp
2     private:
3         int x,y;
4     public:
5         Vector(): x(0), y(0){};
6         Vector(int X,int Y): x(X), y(Y){}
7         void print(){cout << x << "," << y << endl;}
8         Vector operator+(Vector param){
9             Vector temp; // (0,0)
10            temp.x = x + param.x;
11            temp.y = y + param.y;
12            return (temp);
13        }
14    };
15
16 int main(){
17     Vector a (5,2);
18     Vector b (3,1);
19     Vector c;
20     c = a + b;
21     c.print();
22     return 0;
23 }
```



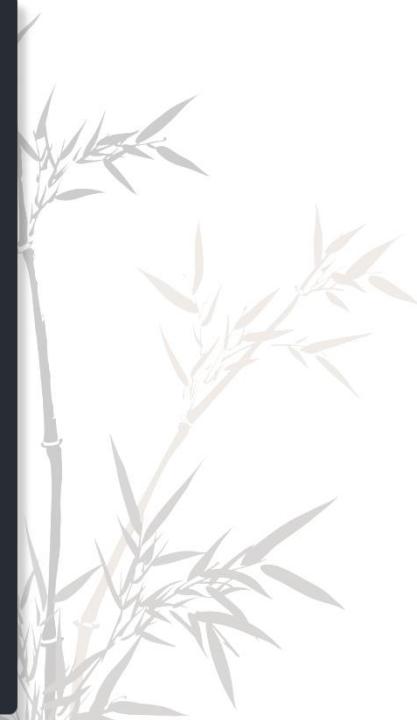
```
~/2023-RobotLab-Cpp-Training
└─./OpOverload
    8,3
```



Polymorphism

- ◆ 讓同名的成員函式，在衍伸類別中能根據類別能夠有不同的行為(貓、狗都會叫，但有不同的叫法)
- ◆ 基礎類別的指針可以指到任意的衍生類別，資料型態是相容的(type-compatible)

```
1 class Polygon {  
2     protected:  
3         int width, height;  
4     public:  
5         void set_values (int a, int b){width=a; height=b;}  
6     };  
7  
8 class Rectangle: public Polygon {  
9     public:  
10        int area (){return (width * height);}  
11    };  
12  
13 class Triangle: public Polygon {  
14     public:  
15        int area (){return (width * height / 2);}  
16    };
```



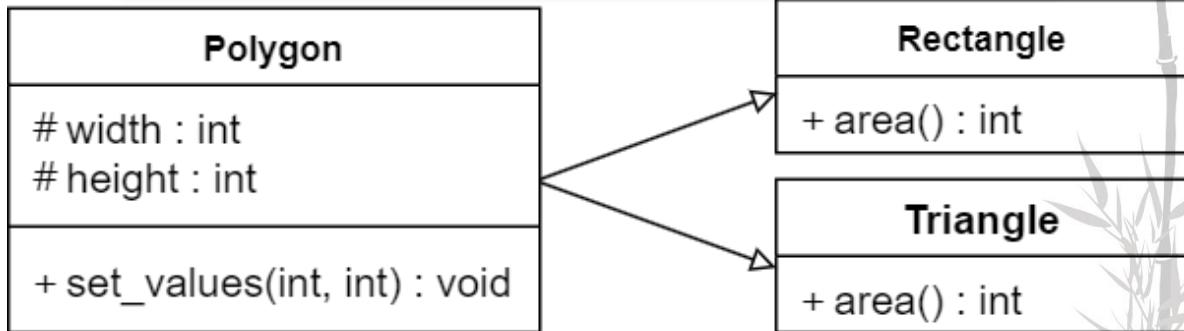


Polymorphism

- ◆ Polygon指針指到衍伸類別無法直接呼叫area()函式，因為Polygon並沒有該成員

Polymorphism-1.cpp

```
1 Rectangle rect;
2 Triangle trgl;
3 Polygon * ppol1 = &rect;
4 Polygon * ppol2 = &trgl;
5 ppol1->set_values (4,5);
6 ppol2->set_values (4,5);
7
8 cout << rect.area() << endl;
9 cout << ppol1->area() << endl; //error
10 cout << trgl.area() << endl;
```

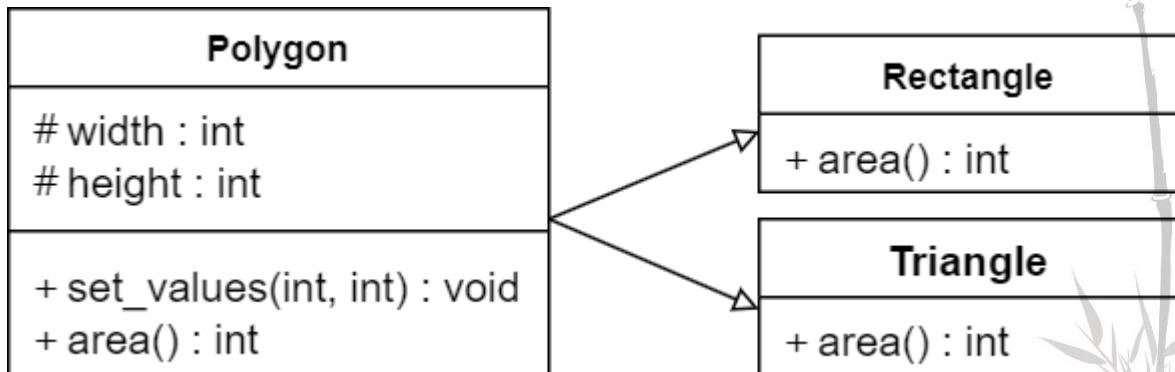




Polymorphism

- ◆ 我們可以宣告area()為虛擬(virtual)函式，讓衍伸類別覆寫(Override)基礎類別的area()，後續覆寫的函式也皆為虛擬

```
1 class Polygon {  
2     protected:  
3         int width, height;  
4     public:  
5         void set_values (int a, int b){width=a; height=b;}  
6         virtual int area(){return -1;}  
7 };
```





Polymorphism

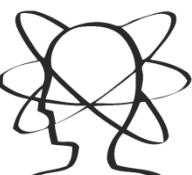
◆ 成功讓area()多型

Polymorphism-2.cpp

```
1 Rectangle rect;
2 Triangle trgl;
3 Polygon poly;
4 Polygon * ppoly1 = &rect;
5 Polygon * ppoly2 = &trgl;
6 Polygon * ppoly3 = &poly;
7 ppoly1->set_values (4,5);
8 ppoly2->set_values (4,5);
9 ppoly3->set_values (4,5);
10 cout << ppoly1->area() << endl;
11 cout << ppoly2->area() << endl;
12 cout << ppoly3->area() << endl;
```

```
~/2023-RobotLab-Cpp-Training
./Polymorphism-2
20
10
-1
```

◆ 如果類別中有任一純虛擬函式，將成為抽象類別 (Abstract Class)，該類別無法實例化(產生物件)



```
1 virtual int area()= 0;
```



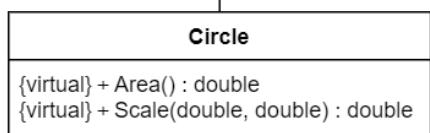
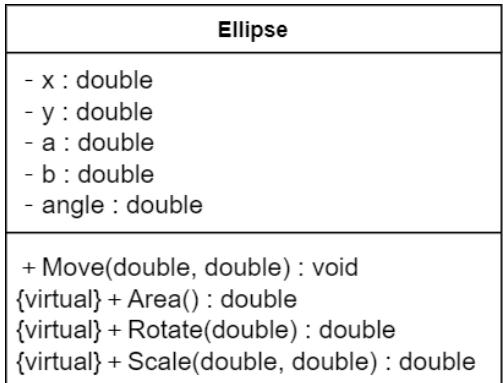
ABC Design

◆ Abstract Base Class(ABC) 抽象基礎類型

- 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

```
1 class Ellipse{
2     private:
3         double x; // x-coordinate of the ellipse's center
4         double y; // y-coordinate of the ellipse's center
5         double a; // semimajor axis
6         double b; // semiminor axis
7         double angle; // orientation angle in degrees
8     public:
9         Ellipse(double X, double Y, double A, double B, double Angle):
10            x(X), y(Y), a(A), b(B), angle(Angle){}
11         void Move(double nx, double ny){x = nx; y = ny;} //nx, ny = new x, new y
12         virtual double Area() {return 3.14159 * a * b;}
13         virtual void Rotate(double dang) {angle += dang;}
14         virtual void Scale(double sa, double sb) {a *= sa; b *= sb;}
15    };
16
17 class Circle: public Ellipse{
18     private:
19         // all needed attributes in Ellipse
20     public:
21         // see a as radius
22         Circle(double X, double Y, double R): Ellipse(X, Y, R, 0, 0){}
23         virtual double Area() {return 3.14159 * a * a;}
24         virtual void Scale(double sa) {a *= sa;}
25    };

```



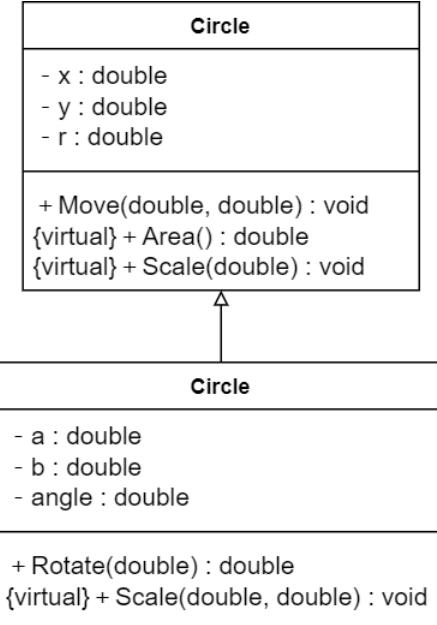
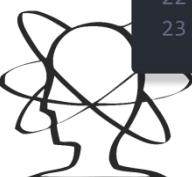
- 1. 成員重複利用效率低
- 2. 屬性(資料成員)語意不易懂



ABC Design

- ◆ 這樣繼承出來的Circle多了很多不必要的屬性，有沒有更好的寫法？

```
1 class Circle{
2     private:
3         double x; // x-coordinate of the circle's center
4         double y; // y-coordinate of the circle's center
5         double r; // radius of circle
6     public:
7         Circle(double X, double Y, double R): x(X), y(Y), r(R){}
8         void Move(int nx, int ny){x = nx; y = ny;}
9         virtual double Area() {return 3.14159 * r * r;}
10        virtual void Scale(double sr) {r *= sr;}
11    };
12
13 class Ellipse: public Circle{
14     private:
15         double a;
16         double b;
17         double angle;
18     public:
19         Ellipse(double X, double Y, double A, double B, double Angle):
20             Circle(X, Y, 0), a(A), b(B), angle(Angle){}
21         void Rotate(double dang) {angle += dang;}
22         virtual void Scale(double sa, double sb) {a *= sa; b *= sb;}
23    };
```

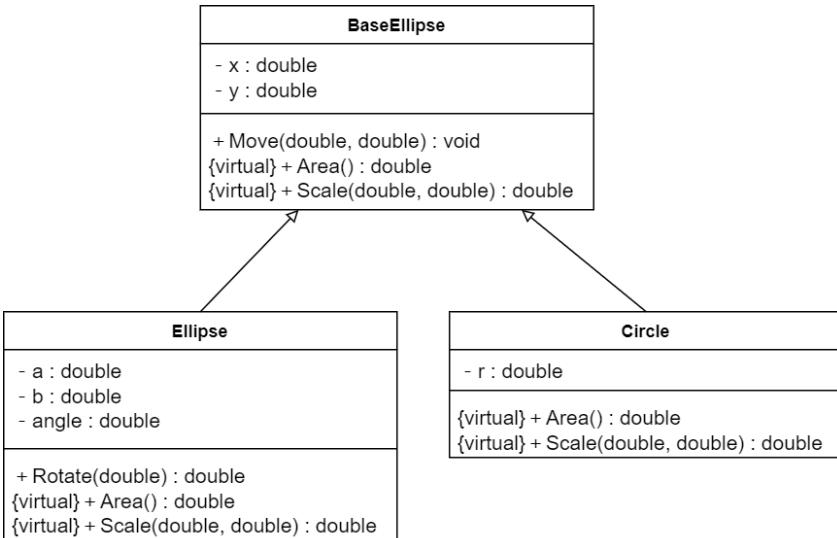




ABC Design

◆ 抽出兩者的共同點作為基礎類別

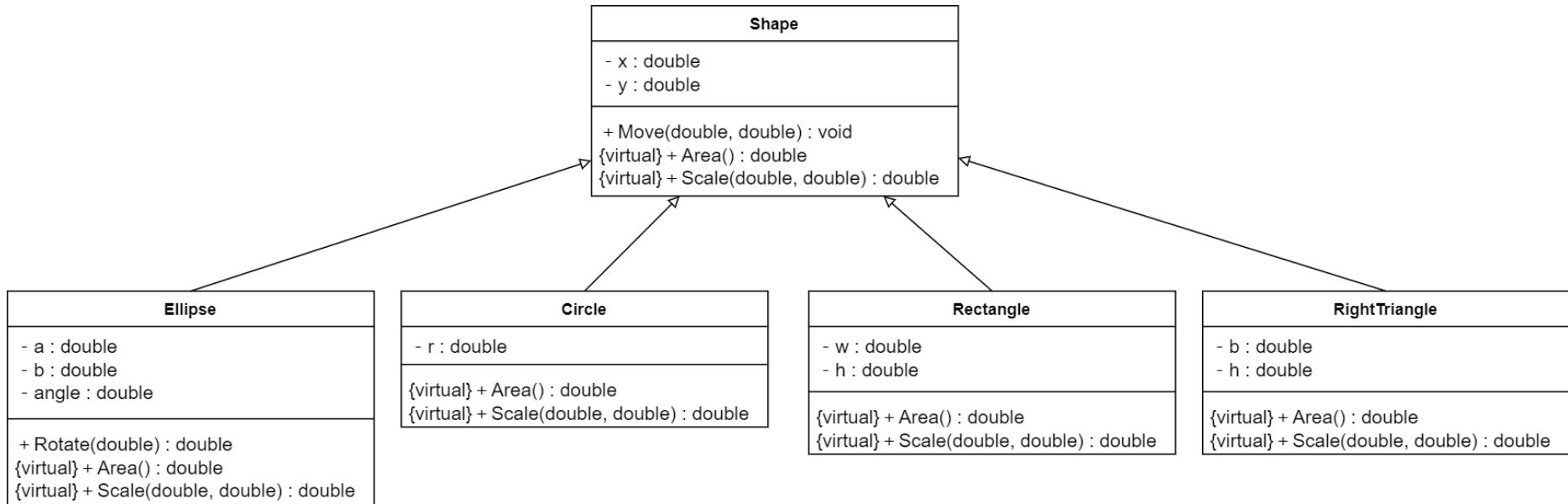
```
1 class BaseEllipse{
2     private:
3         double x;
4         double y;
5     public:
6         BaseEllipse(double X, double Y): x(X), y(Y){}
7         void Move(int nx, int ny) {x = nx; y = ny;}
8         virtual double Area() const = 0;
9         virtual void Scale() = 0;
10    };
11 class Ellipse: public BaseEllipse{
12     private:
13         double a;
14         double b;
15         double angle;
16     public:
17         Ellipse(double X, double Y, double A, double B, double Angle):
18             BaseEllipse(X, Y), a(A), b(B), angle(Angle){}
19         void Rotate(double dang) {angle += dang;}
20         virtual double Area() {return 3.14159 * a * b;}
21         virtual void Scale() {a *= sa; b *= sb;}
22    };
23 class Circle: public BaseEllipse{
24     private:
25         double r;
26     public:
27         Circle(double X, double Y, double R): BaseEllipse(X, Y), r(R){}
28         virtual double Area() {return 3.14159 * r * r;}
29         virtual void Scale() {r *= sr;}
30    };
```





ABC Design

◆ 較靈活且擴充性佳





Appendix

- ◆ All code on github [[Link](#)]

