

變數與輸入及運算

林靖紳

Contents

0 前情提要

1 物件概念

2 變數宣告

3 輸入 Input

4 算術運算子

前情提要

00

C++ 程式架構

最基本的架構

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6
7      // 主要程式碼，像是：
8
9      cout<<"Hello world"<<endl;
10     return 0;
11 }
12
```

C++ 基本概念：

- 程式碼是由上而下執行的
- 在 main 裡面直到遇到 return 0；程式會立刻結束
- 程式中用到任何物件都需要進行宣告 **[本章]**

Output

輸出文字

- 格式: `cout << 變數或是字串1<<變數或是字串2<<變數或是字串3<<....<<變數或是字串n` ;
- 如果要輸出字串(一行文字), 需要利用 “ ” 來框起來, 像是:
 - `cout << “Hello World!” << “Hi” ;`

```
PS C:\Users\tiffany\Desktop> ./week1_test.exe
Hello World!Hi
```

小提醒:

- 輸出文字盡量還是使用 英文 , 中文很常因為環境問題造成編譯問題

- 而 `endl` 表示 end of line, 結束一行, 輸出後換行, 像是:
 - `Cout << “Hello World!” <<endl<< “Hi” <<endl;`

```
PS C:\Users\tiffany\Desktop> ./week1_test.exe
Hello World!
Hi
```

Output

輸出文字

- 跳脫字元
 - 指的是脫離原本字元的意思
 - 加上 \ 用以跳脫，有的字元本身在程式碼中有特殊的用途，需要用 \ 跳脫；
 - 有些則是加上 \ 之後會有脫離原本字元意思的用途
 - 例如前頁的例子中，「”」用來當成 C++ 字串的開頭或是結尾
 - 因此我們需要加上「\」變成「\"」就能在字串中當作純符號來看待
- 試試看 `cout << “\” << endl;`

練習題

跳脫字元

跳脫字元

時間限制 1 秒

跳脫字元指的是脫離原字元的意思，有些字元有原本的意義，所以必須使用跳脫字元，例如以下文字：

```
|'o'|  
\\^_^/  
("o")  
[-%-]
```

就有幾處必須使用跳脫字元。

[輸入說明]

本題無輸入

[輸出說明]

請輸出以下文字：

```
|'o'|  
\\^_^/  
("o")  
[-%-]
```

練習題

跳脫字元

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      cout<<"|\\'o\\'|\\n\\\\^_\\^/\\n(\\'o\\')\\n[-%-]"<<endl;
7      return 0;
8  }
```


C++ 程式架構

最基本的架構

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6
7      // 主要程式碼，像是：
8
9      cout<<"Hello world"<<endl;
10     return 0;
11 }
12
```

C++ 基本概念：

- 程式碼是由上而下執行的
- 在 main 裡面直到遇到 return 0；程式會立刻結束
- 程式中用到任何物件都需要進行宣告 **[本章]**

物件概念

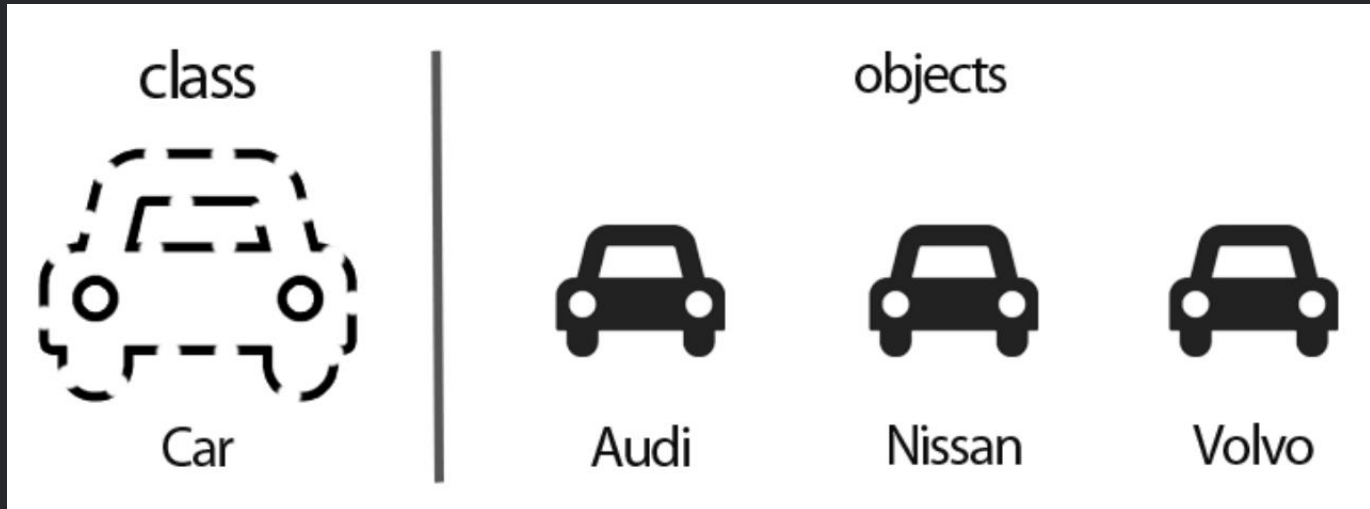
01

物件的概念

- 小總結
 - 物件是甚麼?
 - 可以是變數或是函式
 - 物件導向程式設計的概念
 - 就是將物件們用一個更大的東西—class 包起來
 - Class 之間的關係
 - Inheritance
 - Override
 - 目的: 方便程式設計師組織和管理程式碼、梳理程式設計的思路。特別是對於中大型的程碼提高軟體開發的效率
 - C++, Java, Python, C#, PHP, ... 都有支援

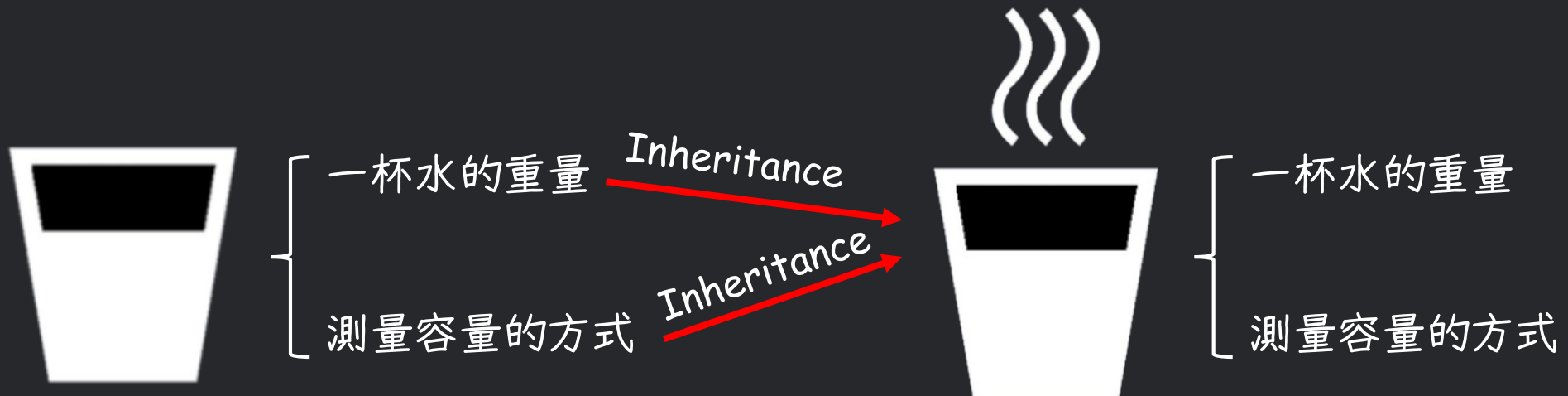
淺談物件導向程式設計的概念

- 物件導向程式設計：
 - Object Oriented Programming, OOP
 - 他是一種概念，期望程式設計者在設計程式時可以朝著：
 - 將相關的變數或是函式 包成一個大的項目，讓使用者透過這個 類別 (class) 去操作裡面的物件 (objects)



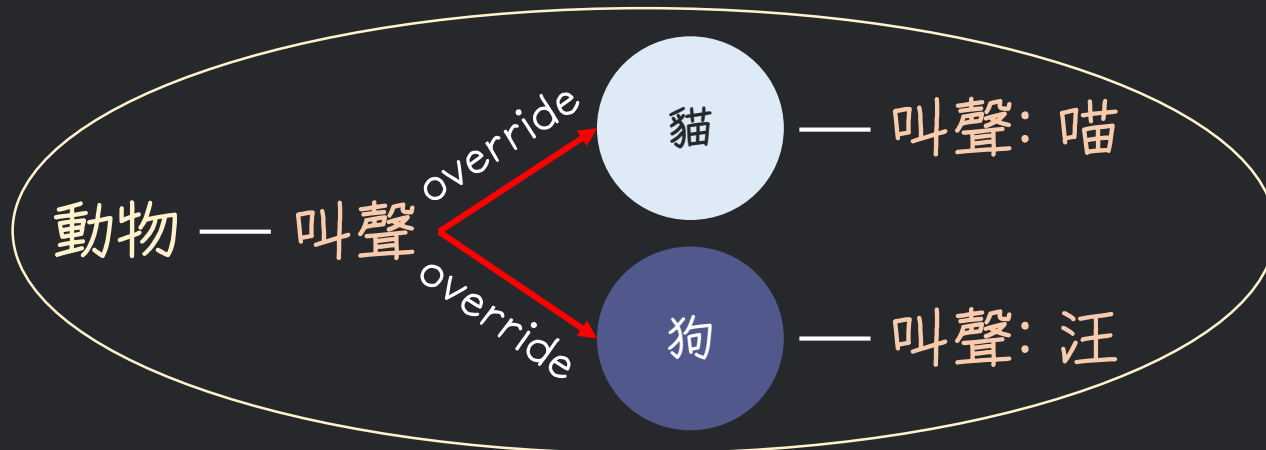
淺談物件導向程式設計的概念

- 物件導向程式設計：
 - Object Oriented Programming, OOP
 - 他是一種概念，期望程式設計者在設計程式時可以朝著：
 - 當兩種不同的類別擁有強烈的歸屬關係時，例如：「水」跟「熱水」這兩個類別。此時「熱水」可以透過「繼承」這個方式去承接「水」那邊的內容物，而不用自己再重寫一次。



淺談物件導向程式設計的概念

- 物件導向程式設計：
 - Object Oriented Programming, OOP
 - 他是一種概念，期望程式設計者在設計程式時可以朝著：
 - 當多種不同的類別有共同的歸屬關係時，例如：「貓」跟「狗」都是「動物」這個類別。
「動物」都有「叫聲」，此時貓跟狗可以透過「覆寫 override」的方式去分別呈現他們各自的「叫聲」。



物件的概念

- 小總結
 - 物件是甚麼?
 - 可以是變數或是函式
 - 物件導向程式設計的概念
 - 就是將物件們用一個更大的東西—class 包起來
 - Class 之間的關係
 - Inheritance
 - Override
 - 目的: 方便程式設計師組織和管理程式碼、梳理程式設計的思路。特別是對於中大型的程碼提高軟體開發的效率
 - C++, Java, Python, C#, PHP, ... 都有支援

變數宣告

02

變數 (Variable)

- 概念
 - 是一個存放資訊的位址 (跟數學上對於變數的定義不太一樣)
 - 資訊
 - 名稱:
 - 識別字 (Identifier, 簡稱: ID), 變數的名稱
 - 資料: (值)
 - 可能是一個整數、小數(浮點數)、字母、符號、字串, ……
 - 位址
 - 記憶體位址

變數 (Variable)

- 概念
 - 是一個存放資訊的位址
 - 資訊
 - 名稱:
 - 識別字 (Identifier, 簡稱: ID), 變數的名稱
 - 資料: (值)
 - 可能是一個整數、小數(浮點數)、字母、符號、字串, ……
- 位址
 - 記憶體位址

變數 (Variable)

- 變數的名稱 (ID)
 - 命名規則:
 - 不可以跟 C++ 標準函數庫中的 Keywords 相同, 像是: return, cout, endl, if, else, ...
 - 英文命名
 - 除了_ 不要有其他特殊符號
 - 變數名稱盡量和內容相關

變數 (Variable)

- 概念
 - 是一個存放資訊的位址
 - 資訊
 - 名稱:
 - 識別字 (Identifier, 簡稱: ID), 變數的名稱
 - 資料: (值)
 - 可能是一個整數、小數(浮點數)、字母、符號、字串, ……
- 位址
 - 記憶體位址

變數 (Variable)

- 資料
 - 型態
 - 值

型態	中文意思	英文字義	位元組 (byte)	可儲存的值 舉例	範圍
int	整數	Integer	4	100, -5, 123456, 0, ...	-2147483648 ~ 2147483647
float	單精度 浮點數	Floating point	4	3.1415, 4.3, -3.3, ... (小數點後六位)	$10^{-38} \sim 10^{38}$
double	雙精度 浮點數	Double floating point	8	3.1415926535, -3.512345678, ... (小數點後十五位)	$10^{-308} \sim 10^{308}$
long long	長整數		8	同 int	-9223372036854775808 ~ 9223372036854775807
char	字元	Character	1	a, b, ;, ~, *, +, ...	參照 ASCII Code
bool	布林(是非)	Boolean	1	true, false	

The ASCII code

American Standard Code for Information Interchange

ASCII control characters			
DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de línea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters									
DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	
32	20h	space	64	40h	@	96	60h	`	
33	21h	!	65	41h	A	97	61h	a	
34	22h	"	66	42h	B	98	62h	b	
35	23h	#	67	43h	C	99	63h	c	
36	24h	\$	68	44h	D	100	64h	d	
37	25h	%	69	45h	E	101	65h	e	
38	26h	&	70	46h	F	102	66h	f	
39	27h	'	71	47h	G	103	67h	g	
40	28h	(72	48h	H	104	68h	h	
41	29h)	73	49h	I	105	69h	i	
42	2Ah	*	74	4Ah	J	106	6Ah	j	
43	2Bh	+	75	4Bh	K	107	6Bh	k	
44	2Ch	,	76	4Ch	L	108	6Ch	l	
45	2Dh	-	77	4Dh	M	109	6Dh	m	
46	2Eh	.	78	4Eh	N	110	6Eh	n	
47	2Fh	/	79	4Fh	O	111	6Fh	o	
48	30h	0	80	50h	P	112	70h	p	
49	31h	1	81	51h	Q	113	71h	q	
50	32h	2	82	52h	R	114	72h	r	
51	33h	3	83	53h	S	115	73h	s	
52	34h	4	84	54h	T	116	74h	t	
53	35h	5	85	55h	U	117	75h	u	
54	36h	6	86	56h	V	118	76h	v	
55	37h	7	87	57h	W	119	77h	w	
56	38h	8	88	58h	X	120	78h	x	
57	39h	9	89	59h	Y	121	79h	y	
58	3Ah	:	90	5Ah	Z	122	7Ah	z	
59	3Bh	;	91	5Bh	[123	7Bh	{	
60	3Ch	<	92	5Ch	\	124	7Ch		
61	3Dh	=	93	5Dh]	125	7Dh	}	
62	3Eh	>	94	5Eh	^	126	7Eh	~	
63	3Fh	?	95	5Fh	_				

theASCIIcode.com.ar

Extended ASCII characters											
DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	Ł	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	ł	225	E1h	ô
130	82h	é	162	A2h	ó	194	C2h	Ł	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	ł	227	E3h	Õ
132	84h	ä	164	A4h	ñ	196	C4h	Ł	228	E4h	ö
133	85h	à	165	A5h	Ñ	197	C5h	ł	229	E5h	Ö
134	86h	á	166	A6h	ª	198	C6h	Ł	230	E6h	μ
135	87h	ç	167	A7h	º	199	C7h	ł	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	Ł	232	E8h	ƒ
137	89h	ë	169	A9h	®	201	C9h	ł	233	E9h	Ů
138	8Ah	è	170	AAh	¬	202	CAh	Ł	234	EAh	Ú
139	8Bh	ï	171	ABh	½	203	CBh	ł	235	EBh	Û
140	8Ch	ì	172	ACH	¼	204	CCh	Ł	236	ECh	Ü
141	8Dh	í	173	ADh	¾	205	CDh	ł	237	EDh	Ý
142	8Eh	Ä	174	Aeh	«	206	CEh	Ł	238	EEh	Ŷ
143	8Fh	Å	175	Afh	»	207	CFh	ł	239	EFh	·
144	90h	É	176	B0h	⋮	208	D0h	Ł	240	F0h	±
145	91h	æ	177	B1h	⋮	209	D1h	ł	241	F1h	±
146	92h	Æ	178	B2h	⋮	210	D2h	Ł	242	F2h	̄
147	93h	ô	179	B3h	⋮	211	D3h	ł	243	F3h	¾
148	94h	ò	180	B4h	⋮	212	D4h	Ł	244	F4h	¶
149	95h	ó	181	B5h	⋮	213	D5h	ł	245	F5h	§
150	96h	û	182	B6h	⋮	214	D6h	Ł	246	F6h	÷
151	97h	ù	183	B7h	⋮	215	D7h	ł	247	F7h	º
152	98h	ÿ	184	B8h	©	216	D8h	Ł	248	F8h	ˆ
153	99h	Ö	185	B9h	ƒ	217	D9h	ł	249	F9h	˙
154	9Ah	Ü	186	BAh	ƒ	218	DAh	Ł	250	FAh	˙
155	9Bh	ø	187	BBh	ƒ	219	DBh	ł	251	FBh	˙
156	9Ch	£	188	BCb	ƒ	220	DCh	Ł	252	FCh	˙
157	9Dh	Ø	189	BDh	¢	221	DDh	ł	253	FDh	˙
158	9Eh	×	190	BEh	¥	222	DEh	Ł	254	FEh	˙
159	9Fh	ƒ	191	BFh	¬	223	DFh	ł	255	FFh	˙

宣告變數

- 目的
 - 就是去告訴電腦：「我要使用變數囉!」，讓電腦準備一個記憶體位址來存放變數。
- 公式
 - [型態][變數名稱];
 - [型態][變數名稱] = [賦值];

注意:

利用 char 宣告變數，表示存放的資料是字元，需要單引號 ' ' 框起來。
也因此 ' 是跳脫字元哦~

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int integer;
7      float floating_num = 3.5;
8      double double_num;
9      char chr = 'a';
10     long long long_number = 52456;
11
12     return 0;
13 }
```

宣告變數

- 驗證: 加上 cout 輸出看看剛剛宣告的變數, 有沒有把值存進去

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int integer;
7      float floating_num = 3.5;
8      double double_num;
9      char chr = 'a';
10     long long long_number = 52456;
11
12     cout<<"int: "<<integer<<endl;
13     cout<<"float: "<<floating_num<<endl;
14     cout<<"double: "<<double_num<<endl;
15     cout<<"char: "<<chr<<endl;
16     cout<<"long long: "<<long_number<<endl;
17
18     return 0;
19 }
20
```

```
PS C:\Users\user\Desktop> ./test.exe
int: 0
float: 3.5
double: 4.94066e-324
char: a
long long: 52456
```


宣告變數

- 但是發生了奇怪的事情！！

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int integer;
7      float floating_num = 3.5;
8      double double_num;
9      char chr = 'a';
10     long long long_number = 52456;
11
12     cout<<"int: "<<integer<<endl;
13     cout<<"float: "<<floating_num<<endl;
14     cout<<"double: "<<double_num<<endl;
15     cout<<"char: "<<chr<<endl;
16     cout<<"long long: "<<long_number<<endl;
17
18     return 0;
19 }
20
```

```
PS C:\Users\user\Desktop> ./test.exe
int: 0
float: 3.5
double: 4.94066e-324
char: a
long long: 52456
```

宣告變數

- 你會發現，就算 `int integer;` 沒有賦值，我們輸出出來竟然會是 0！
- 在宣告時賦值，我們稱為初始化
- 電腦會將宣告時沒有初始化的變數，進行初始化。
- 雖然常常會是歸 0；但也會遇到電腦初始化到其他地方的時候，像是我們範例中宣告的 `double`
- 因此要盡量 養成宣告變數時就賦值、初始化的好習慣！
 - 主要是怕宣告的變數沒有做任何的處理就直接拿來用

```
PS C:\Users\user\Desktop> ./test.exe
int: 0
float: 3.5
double: 4.94066e-324
char: a
long long: 52456
```

宣告變數

- 一次宣告同一個型態的多個變數
 - [型態][變數名稱1], [變數名稱2], [變數名稱3], ... [變數名稱 n]
- 舉例:
 - `int num1=0, num2=0, num3=5;`
 - `float f1 = 3.2, f2 = 2.5, f3=0.0;`
 - `char c1 = 'a' , c2= 'b' , c3= ' ' ;`

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int num1=0, num2=0, num3=5;
7      float f1 = 3.2, f2 = 2.5, f3=2.5;
8      char c1 = 'a', c2 = 'b', c3=' ' ;
9
10     return 0;
11 }
```

宣告變數

- 驗證: 加上 cout 輸出看看剛剛宣告的變數, 有沒有把值存進去

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int num1=0, num2=0, num3=5;
7      float f1 = 3.2, f2 = 2.5, f3=2.5;
8      char c1 = 'a', c2 = 'b', c3=' ';
9      cout<<"num1= "<<num1<<"\t"<<"num2= "<<num2<<"\t"<<"num3= "<<num3<<endl;
10     cout<<"f1= "<<f1<<"\t"<<"f2= "<<f2<<"\t"<<"f3= "<<f3<<endl;
11     cout<<"c1= "<<c1<<"\t"<<"c2= "<<c2<<"\t"<<"c3= "<<c3<<endl;
12     return 0;
13 }
```

宣告變數

- 驗證: 加上 cout 輸出看看剛剛宣告的變數, 有沒有把值存進去

```
PS C:\Users\user\Desktop> g++ test.cpp -g -o test.exe
PS C:\Users\user\Desktop> ./test.exe
num1= 0,          num2= 0,          num3= 5
f1= 3.2,          f2= 2.5,          f3= 0
c1= a,  c2= b,  c3=
```

宣告變數

- 思考問題
 - 請觀察下面的程式碼，先猜猜看結果是甚麼呢？

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int numA = 60;
6      int numB = 80;
7
8      numA = numB;
9      numB = numA;
10
11     cout << numA << " " << numB << endl;
12
13     return 0;
14 }
```

宣告變數

- 指派變數值
 - [變數名稱] = [新的儲存值/運算式];
 - 像是
 - $a = 1;$
 - $a = a + 2;$
 - 等號 “=”，在程式裡面是指派 (assign) 的意思，跟數學上的等號意義不一樣!!
 - 左邊變數的儲存值會變成右邊新的儲存值
 - 而舊的變數值就會消失

宣告變數

- 思考問題
 - 再仔細觀察一下，試著解釋看看程式碼

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int numA = 60;
6      int numB = 80;
7
8      numA = numB;
9      numB = numA;
10
11     cout << numA << " " << numB << endl;
12
13     return 0;
14 }
```


宣告變數

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int numA = 60;
6      int numB = 80;
7
8      numA = numB;
9      numB = numA;
10
11     cout << numA << " " << numB << endl;
12
13     return 0;
14 }
```

```
PS C:\Users\user\Desktop> g++ test.cpp
PS C:\Users\user\Desktop> ./test.exe
80 80
```

宣告變數

- 思考問題
 - 那麼我們要怎麼讓兩個變數的值互換呢？
 - 想想看下方程式碼造成兩個變數值不能互換的原因是甚麼？

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int numA = 60;
6      int numB = 80;
7
8      numA = numB;
9      numB = numA;
10
11     cout << numA << " " << numB << endl;
12
13     return 0;
14 }
```

宣告變數

- 思考問題
 - 那麼我們要怎麼讓兩個變數的值互換呢？
 - 想想看下方程式碼造成兩個變數值不能互換的原因是甚麼？

- 60 這個數值會因為指派 新值給 numA 時被覆蓋掉
- 可以想像在更改文件時，將原本的舊檔覆蓋，造成找不回舊檔
 - 這種時候我們通常會做甚麼呢？
→ 備份檔案

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int numA = 60;
6      int numB = 80;
7
8      numA = numB;
9      numB = numA;
10
11     cout << numA << " " << numB << endl;
12
13     return 0;
14 }
```

宣告變數

- 思考問題
 - 那麼我們要怎麼讓兩個變數的值互換呢？

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5      int numA = 60;
6      int numB = 80;
7      int tmp = numA;
8      numA = numB;
9      numB = tmp;
10
11      cout << numA << " " << numB << endl;
12
13      return 0;
14 }
```

輸入 Input

03

Input

- 從鍵盤輸入文字
 - 格式: `cin>>變數名稱1>>變數名稱2>>變數名稱3>>...>>變數名稱 n;`
 - 執行時, 會看到 █ 閃爍, 等待輸入
 - C++會幫我們略過輸入資料之間的空白、換行。

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int num;
7      cin>>num;
8      return 0;
9  }
```

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int num;
7      float f1;
8      cin>>num>>f1;
9      return 0;
10 }
```

Input

- cin 完後，放上 cout 看看

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main(){
6      int num;
7      float f1;
8      cin>>num>>f1;
9      cout<<"num= "<<num<<endl;
10     cout<<"f1= "<<f1<<endl;
11     return 0;
12 }
```

```
PS C:\Users\user\Desktop> g++ test.cpp -g -o test.exe
PS C:\Users\user\Desktop> ./test.exe
12
2.5
num= 12
f1= 2.5
PS C:\Users\user\Desktop> █
```

[練習A]

算術運算子

04

算術運算子

- 算術運算子 (arithmetic operator)
 - 其實就是我們數學上常用的用算符號: 加、減、乘、除、取餘數
 - 需要兩個運算元 (operands) 去構成運算式

算術運算子	代表意義
$A + B$	加
$A - B$	減
$A * B$	乘
A / B	除
$A \% B$	取餘數 (Mod, 取模) (注意: A、B 必須為整數型態)

算術運算子

- 範例:

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main() {
7      int a=99, b=36, c=0;
8      c = a + b;
9      cout << "a + b = " << c << endl;
10
11     c = a - b;
12     cout << "a - b = " << c << endl;
13
14     c = a * b;
15     cout << "a * b = " << c << endl;
16
17     c = a / b;
18     cout << "a / b = " << c << endl;
19
20     c = a % b;
21     cout << "a % b = " << c << endl;
22
23     return 0;
24 }
```

```
PS C:\Users\user\Desktop> ./test.exe
a + b = 135
a - b = 63
a * b = 3564
a / b = 2
a % b = 27
```

[練習B]

[練習C]

算術運算子

- 範例:

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  int main() {
7      int a=99, b=36, c=0;
8      c = a + b;
9      cout << "a + b = " << c << endl;
10
11     c = a - b;
12     cout << "a - b = " << c << endl;
13
14     c = a * b;
15     cout << "a * b = " << c << endl;
16
17     c = a / b;
18     cout << "a / b = " << c << endl;
19
20     c = a % b;
21     cout << "a % b = " << c << endl;
22
23     return 0;
24 }
```

```
PS C:\Users\user\Desktop> ./test.exe
a + b = 135
a - b = 63
a * b = 3564
a / b = 2
a % b = 27
```

發現:

當兩整數相除，結果只取商數!

算術運算子


- 這時候應該會有一個疑問：那兩個整數相除，我可以得到小數嗎？
- 試試把 `int c`；改成 `float c`？

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int a = 99;
7      int b = 36;
8      int c = a/b;
9      cout<<c<<endl;
10     return 0;
11 }
```



```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int a = 99;
7      int b = 36;
8      float c = a/b;
9      cout<<c<<endl;
10     return 0;
11 }
```

算術運算子

- 這時候應該會有一個疑問：那兩個整數相除，我可以得到小數嗎？
- 試試把 `int c;` 改成 `float c;` ?  發現還是 2

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int a = 99;
7      int b = 36;
8      float c = a/b;
9      cout<<c<<endl;
10     return 0;
11 }
```

```
PS C:\Users\user\Desktop> g++ test.cpp
PS C:\Users\user\Desktop> ./test.exe
2
```

算術運算子

- 這時候應該會有一個疑問：那兩個整數相除，我可以得到小數嗎？
- 答：可以的！
 - 方法一：強制轉型
 - 不是一個好的 coding style
 - 但是很方便
 - 方法二：改變儲存值的型態 [建議]

[練習D]

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int a = 99;
7      int b = 36;
8      float way1 = (float)a/b;
9      cout<<"way1: "<<way1<<endl;
10     float way2 = a/(b+0.0);
11     cout<<"way2: "<<way2<<endl;
12     return 0;
13 }
```

Thank You