Chapter 2
Data Manipulation

Yu-Hsiang Cheng
鄭宇翔 (Slighten)
GOTC Instructor

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

Outline

- Bits and Bytes
- Variables
- Operators
- Some Practices

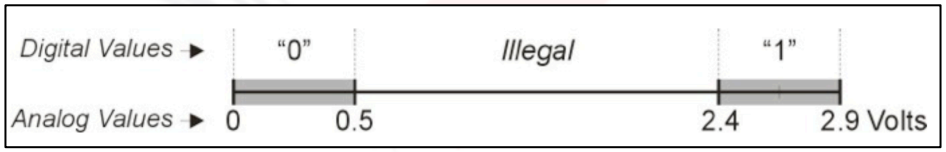
Outline

- Bits and Bytes
- Variables
- Operators
- Some Practices

Bits and Bytes

電腦的基本單位: Bit (位元)

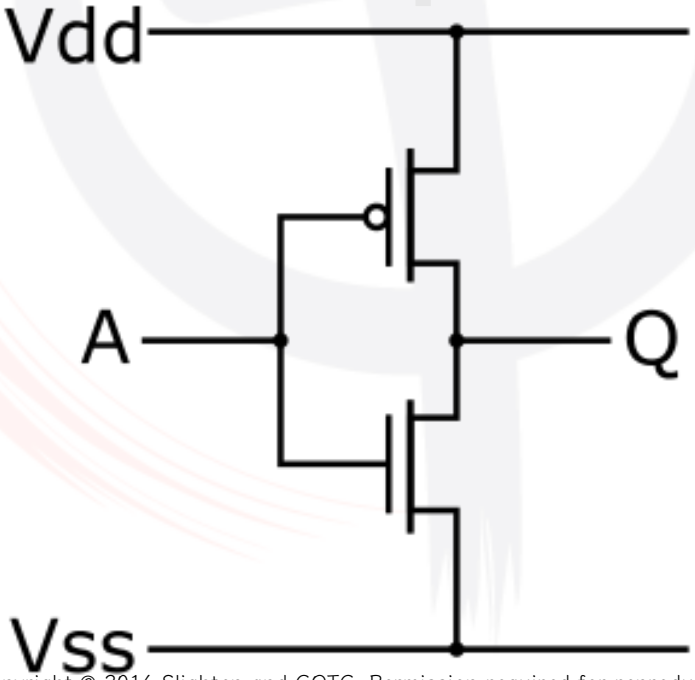
- Bit = Binary digit
- 有電壓 – 1；無電壓 – 0



- 電腦的細胞：電晶體 (transistor)
- 二進位系統 (Binary (base two) system)
- 1 bit → 2 possible states: 0, 1
- 2 bits → 4 possible states: 00, 01, 10, 11
- 3 bits → 8 possible states: 000, 001, 010, 011, 100, 101, 110, 111
- n bits → 2^n possible states

CMOS Transistor

- 電腦的細胞：電晶體 (transistor) – 想成一個開關
- CMOS: **C**omplementary **M**etal–**O**xide–**S**emiconductor (互補式金屬氧化物半導體)
- Inverter (反相器)



我們需要用程式表示與操作哪些現實生活中的資料？

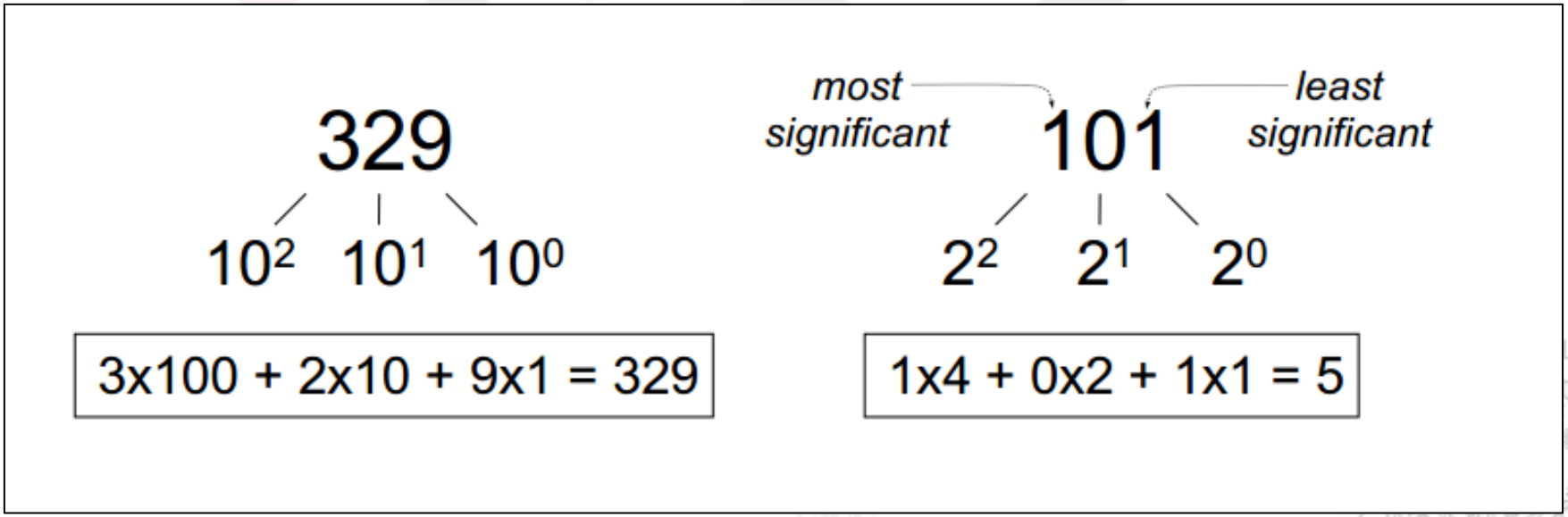
- 數字 (Numbers)
  - 有號 (signed), 無號 (unsigned), 整數 (integers), 浮點數 (floating point), 複數 (complex numbers), 有理數 (rational numbers), 無理數 (irrational numbers), ...
- 文字 (Text)
  - 字元 (characters), 字串 (strings), ...
- 圖片 (Images)
  - 像素 (pixels), 顏色 (colors), 形狀 (shapes), ...
- 聲音 (Sound)
- 邏輯 (Logical)
  - 真 (true), 假 (false)
- 先從數字開始

Terminology

- 進位 ＝ 進制
- Binary: 2 進位、2 進制 (10010101)
- Decimal: 10 進位、10 進制 (9547)
- Octal: 8 進位、8 進制 (0174)
- Hexadecimal: 16 進位、16 進制 (0x2347FACE)

Binary to Decimal (1/2)

- **Non-positional notation: 1, 11, 111, 1111, 11111 (問題?)**
- **Weighted positional notation: 常用的 10 進位**
- MSB: Most Significant Bit (通常最左邊)
- LSB: Least Significant Bit (通常最右邊)
- *unsigned int*: 正整數 (n bits 代表 $0 \sim 2^n - 1$ 的所有數字)



Binary to Decimal (2/2)

- 常用到背起來

2^2	2^1	2^0	decimal
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Binary ⇌ Hexadecimal

- 16 進制： 1 個數字有 16 種可能：0~9, a, b, c, d, e, f
- 2 進制： 4 個 bits 有 $2^4 = 16$ 種可能
- 所以 2 進位的 4 個數字 = 16 進位的 1 個數字
- 從最右邊開始 4 個 1 組，翻成 10 進位，再翻成 16 進位

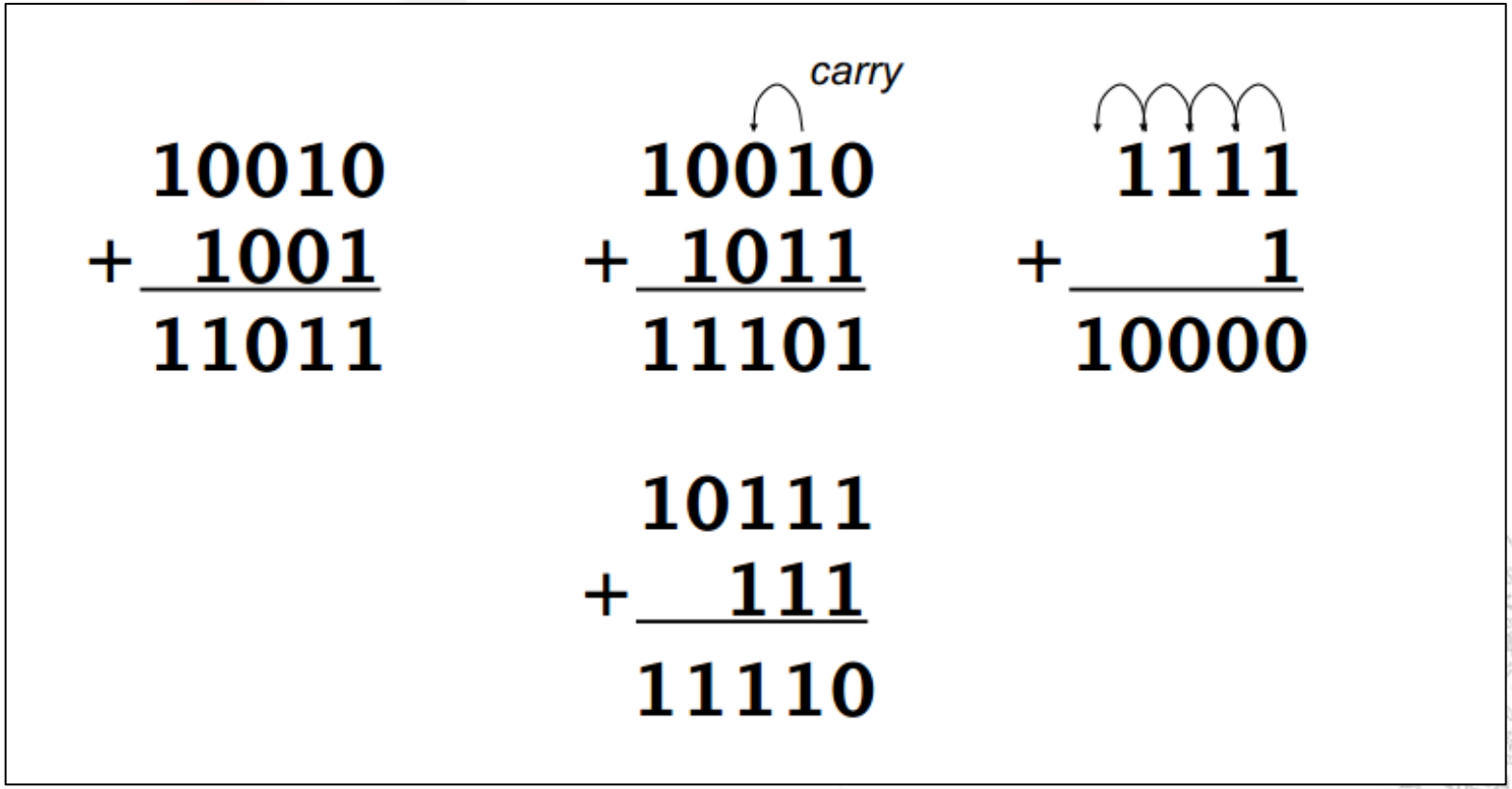
- 10111001010110111001
- 10|1110|0101|1011|1001
- 2| 14 | 5 | 11 | 9
- 2| e | 5 | b | 9
- 0x2e5b9

Arithmetic Operations

- Addition
- Subtraction

Addition/Subtraction

- *unsigned int*: like decimal
- carry: 進位

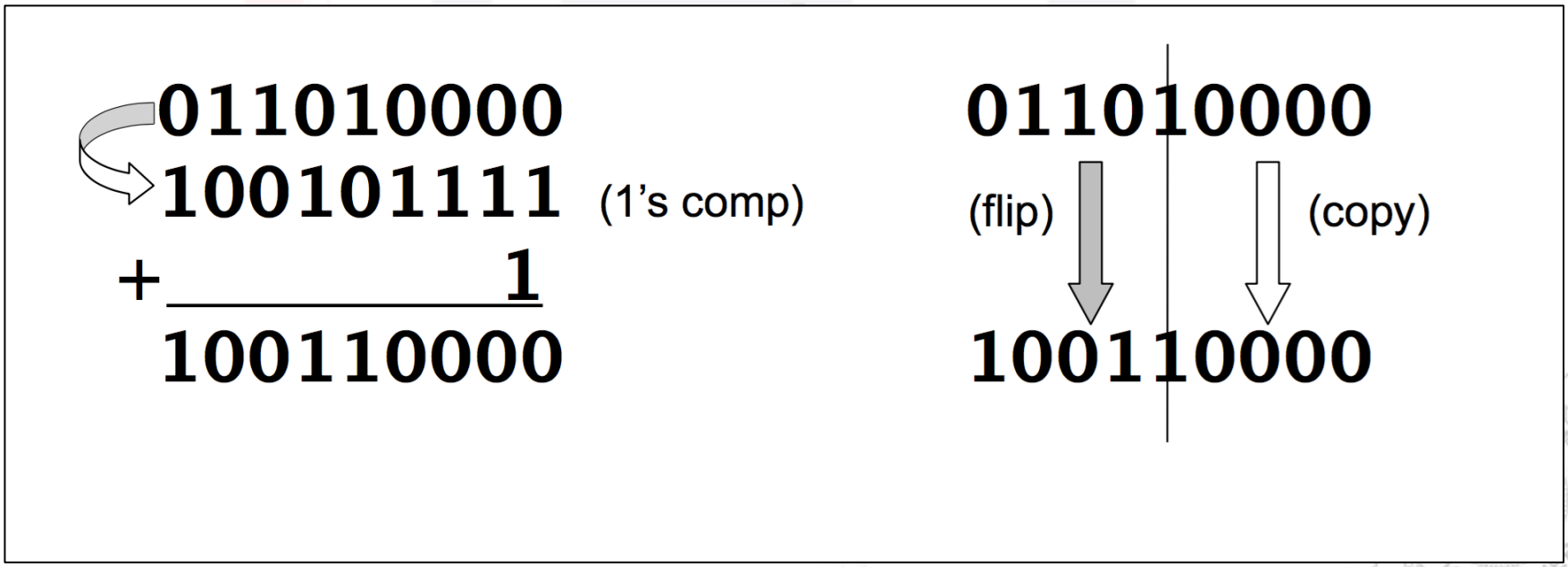


如何表示負數？

- (signed) int: 整數 (有正/負號)
- Q: 如何在電腦中表示負數？
- 符號帶大小 (Sign–Magnitude): MSB 0 = 正, MSB 1 = 負
  - $10101_2 = -5_{10}$
- 1 補數 (1's Complement): 負數=正數的每個 bit 0 變 1、1 變 0
  - $00101_2 \rightarrow 11010_2 = -5_{10}$
  - 已少用
- 2 補數 (2's Complement): 1 補數 + 1
  - $11010_2 + 1 = 11011_2 = -5_{10}$
  - 常用
  - n bits: $-2^{n-1} \sim 2^{n-1} - 1$.

2 補數

- 2 補數 (2's Complement) 兩種方式
  1. 1 補數加 1 (全部翻轉加 1)
  2. 從最右邊往左數到第 1 個 1 之後左邊都翻轉



Comparison Table

- 現今常用 2 補數，因為
  1. 沒有 +0, –0 之分
  2. 硬體上較好實作

+N	Positive integers (for all systems)	-N	Negative integers		
			Sign and magnitude	1's complement $\overline{N}$	2's complement N*
+0	0000	-0	1000	1111	-----
+1	0001	-1	1001	1110	1111
+2	0010	-2	1010	1101	1110
+3	0011	-3	1011	1100	1101
+4	0100	-4	1100	1011	1100
+5	0101	-5	1101	1010	1011
+6	0110	-6	1110	1001	1010
+7	0111	-7	1111	1000	1001
		-8	-----	-----	1000

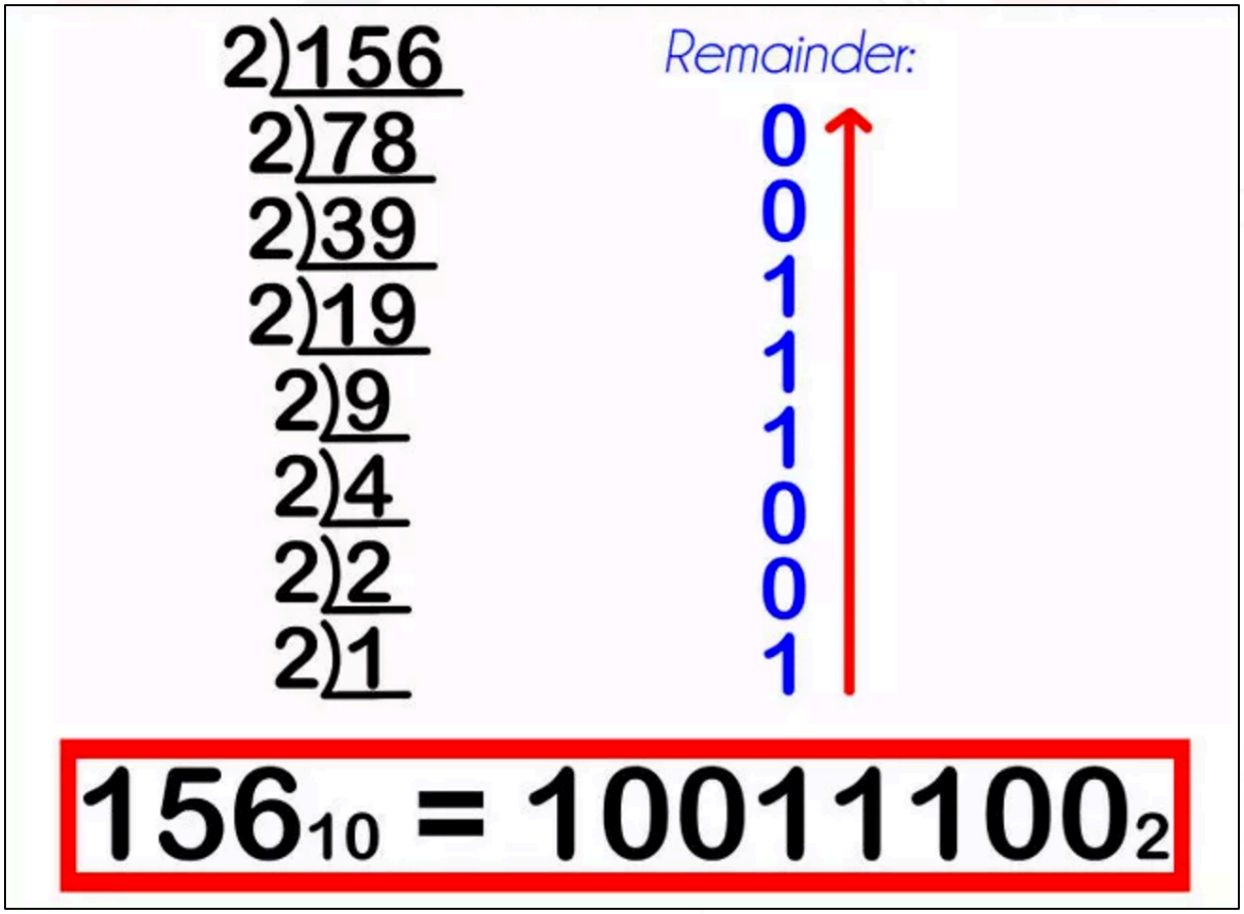
2 補數正負轉換

- $Y = 0111_2 = 7$
- $-Y = \sim Y + 1 = 1000_2 + 1 = 1001_2 = -7$

- $X = 11100110_2 = ?$
- $-X = 00011010_2 = 2^4 + 2^3 + 2^1 = 16 + 8 + 2 = 26$
- 所以 $X = -26$

Decimal to Binary

- 短除法



溢位 (Overflow)

- 發生在 signed integer (有號整數、整數)

- 假設第 4 位是最高位 (MSB)
- 1101 + 0100 = 10001 (truncated) = 0001 (?)
- 13 + 4 = 17 (truncated) = 1 (X)
- –3 + 4 = 1 (O)
- Not overflow!

- 0101 + 0100 = 1001 (?) (正 + 正 = 負)
- 5 + 4 (= 9 (should)) = –7 (overflow!)
- $-2^{4-1} \sim 2^{4-1} - 1 = -8 \sim 7$

- 同理：負 + 負 = 正，也是 overflow!
- 但正 + 負、負 + 正，一定不會 overflow!

Logical Operations

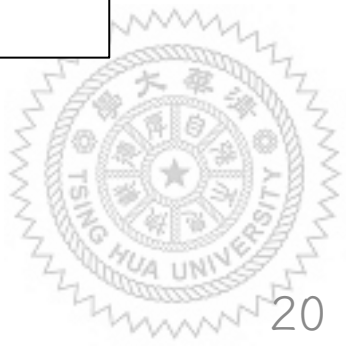
- AND (&)
- OR (|)
- NOT (~)

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A
0	1
1	0

- All bitwise



Examples of Logical Operations

- AND (&)
  - 清除 bits 很好用 (mask)
  - AND with zero = 0
  - AND with one = no change
- OR (|)
  - set bits 很好用
  - OR with zero = no change
  - OR with one = 1
- NOT (~)
  - unary operation –– one argument (~A)
  - flips every bit

```
  11000101
AND 00001111
  00000101

  11000101
OR  00001111
  11001111

NOT 11000101
  00111010
```

Extension

- 將兩個不同長度的 2 進位數字做算數運算，需要把較短的數字延伸
- 有兩種延伸方式: signed or unsigned
- Unsigned (Shift Right Logical): 左邊都補 0

4-bit	8-bit
0100 (4)	00000100 (still 4)
1100 (-4)	00001100 (12, not -4)

- Signed (Shift Right Arithmetic): 最左邊是 1 補 1，是 0 補 0 (保號)

4-bit	8-bit
0100 (4)	00000100 (still 4)
1100 (-4)	11111100 (still -4)

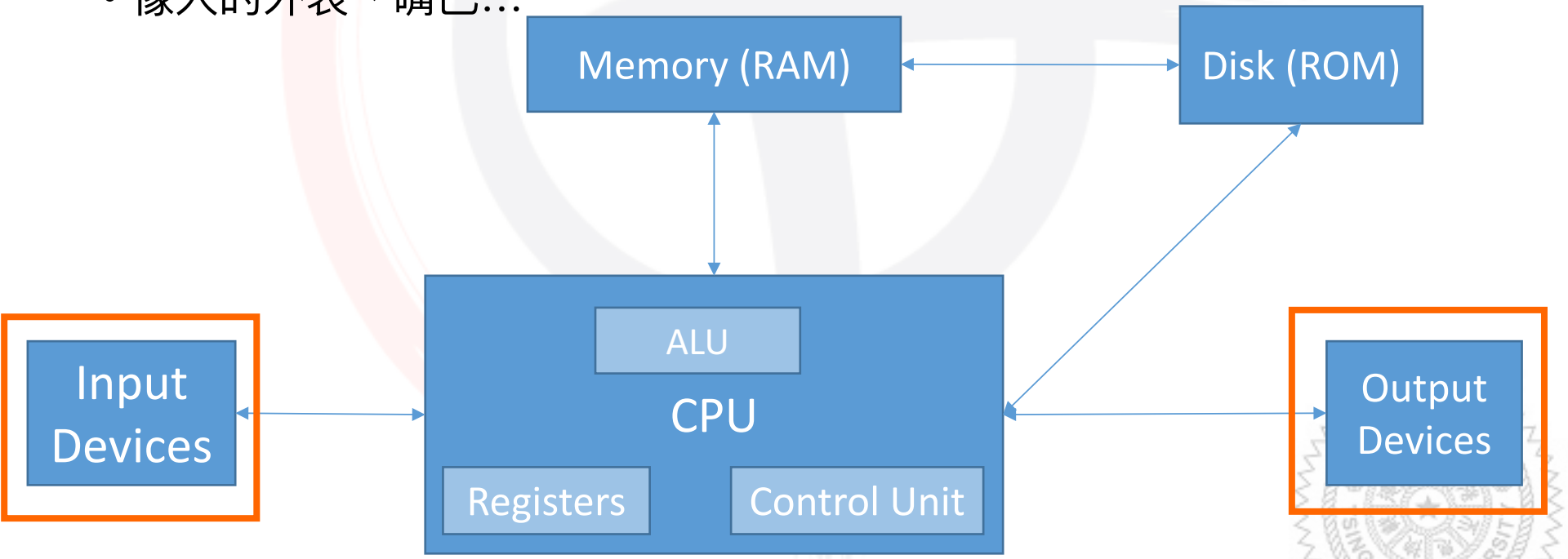
Outline

- Bits and Bytes
- Variables
- Operators
- Some Practices

A Computer Schematic

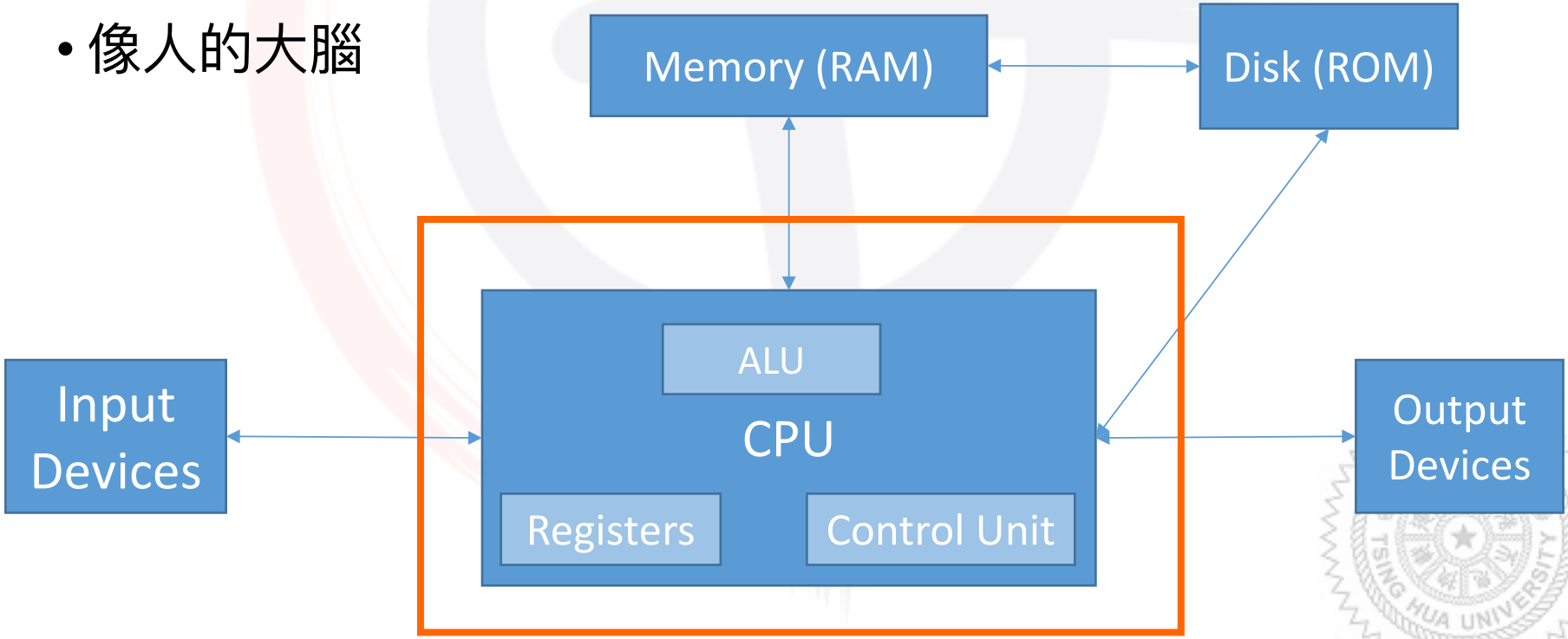
A Computer Schematic (1/5)

- Input Devices: 滑鼠、鍵盤、光碟機…
  - 像人的耳朵、眼睛…
- Output Devices: 螢幕、喇叭…
  - 像人的外表、嘴巴…



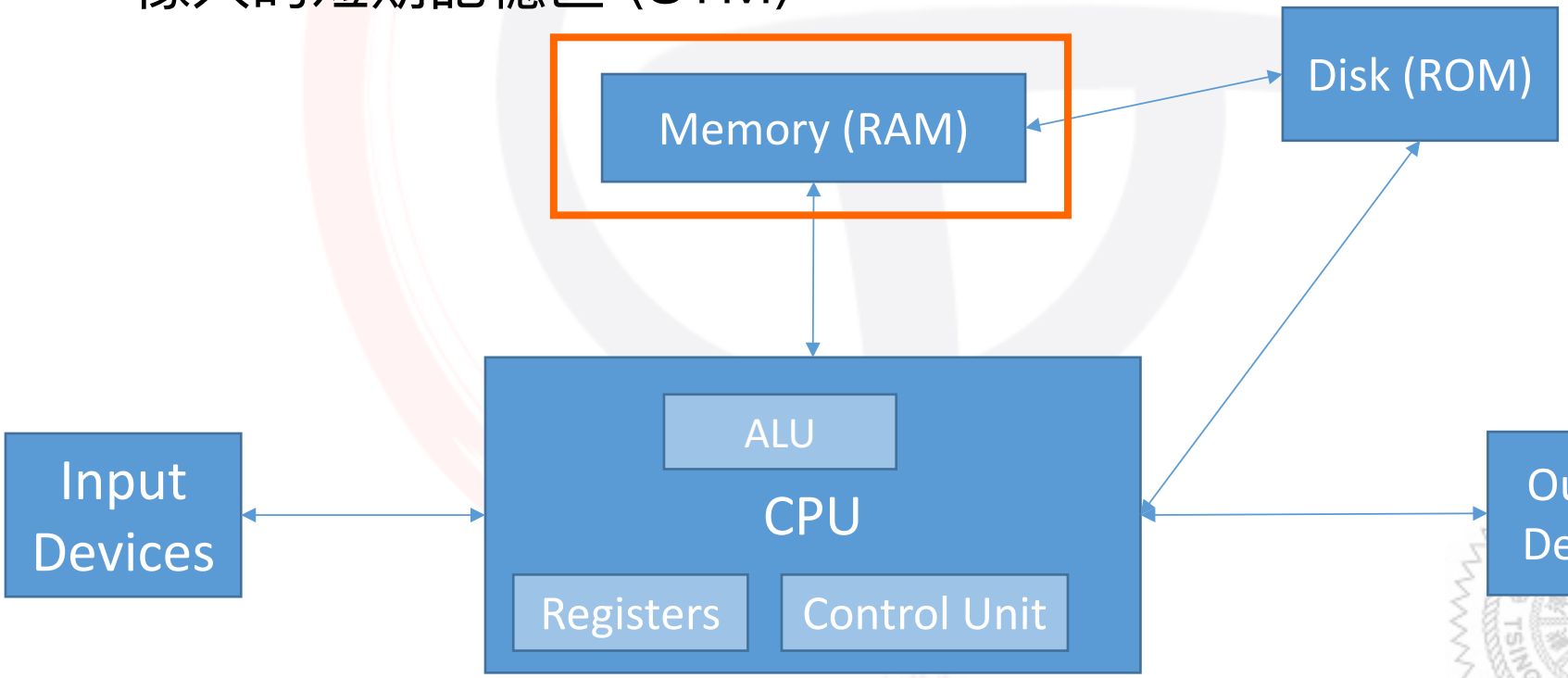
A Computer Schematic (2/5)

- CPU (Central Process Unit, 中央處理單元)：包含 ALU (Arithmetic Logic Unit, 算術邏輯單元)、Registers (暫存器)、Control Unit (控制單元)
  - 執行指令 (instructions)
  - 像人的大腦



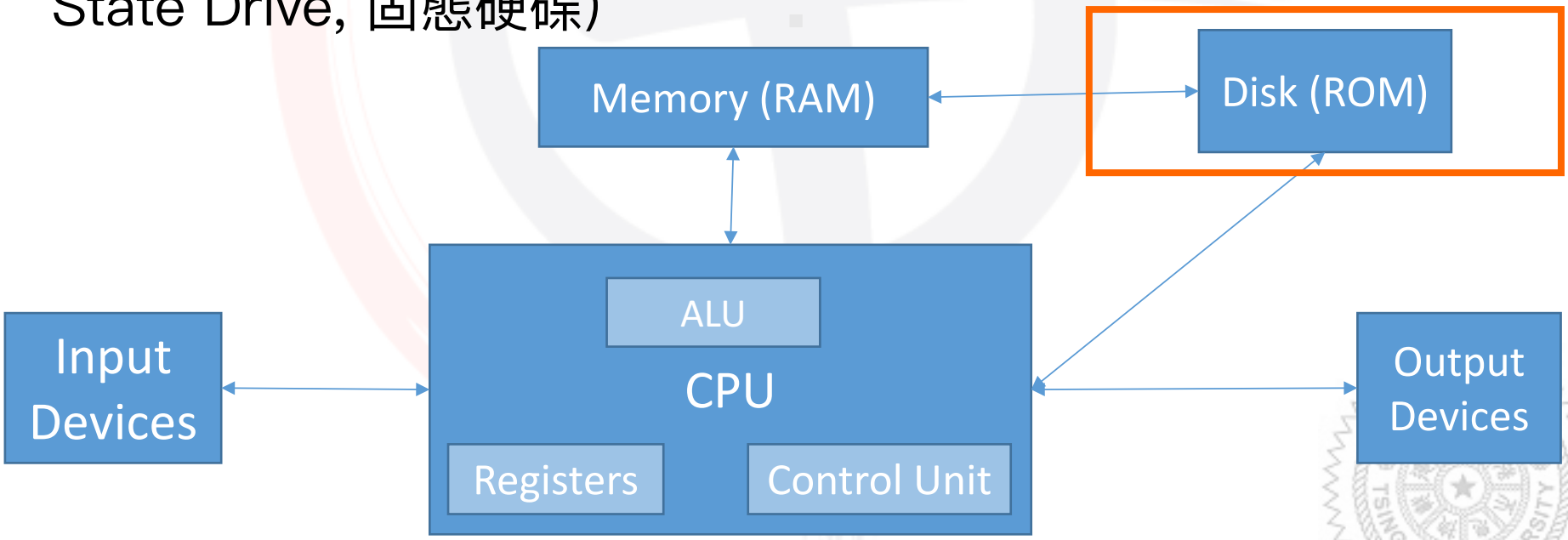
A Computer Schematic (3/5)

- RAM (Random Access Memory, 隨機存取記憶體)
  - Volatile (揮發性的): 沒電就無法保存資料
  - 像人的短期記憶區 (STM)



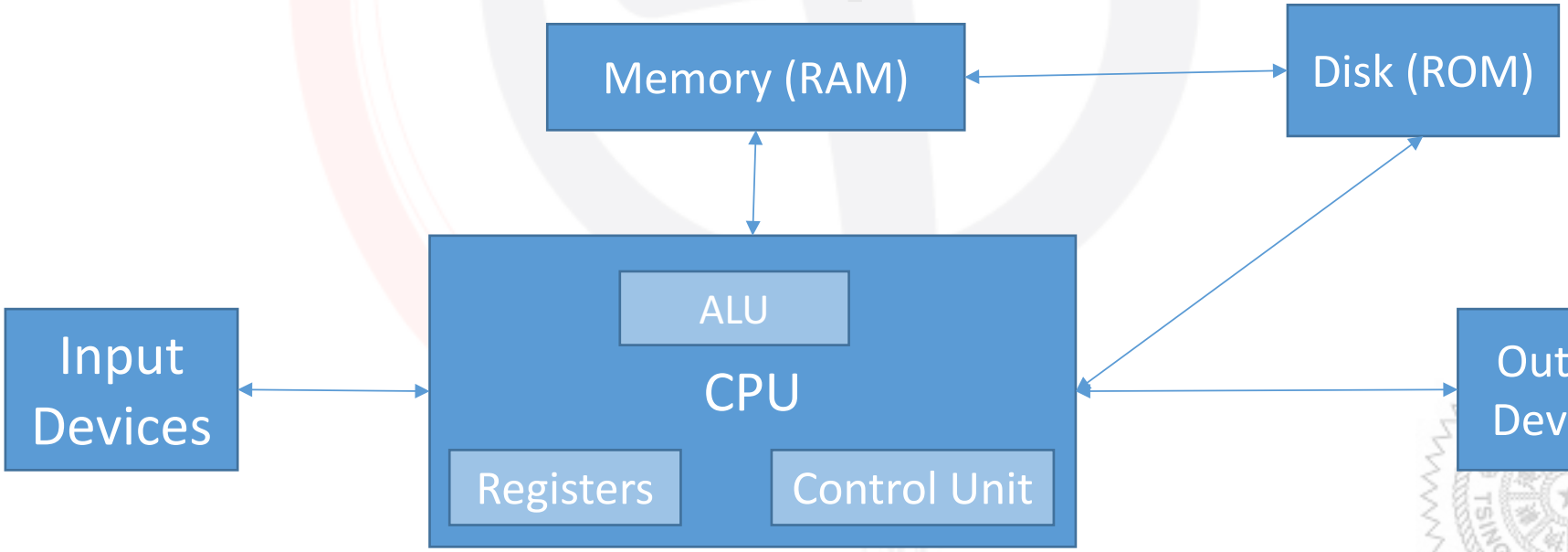
A Computer Schematic (4/5)

- ROM (Read-Only Memory, 唯讀記憶體)
  - Non-volatile (非揮發性的): 沒電仍保存資料
  - 像人的長期記憶區 (LTM)
- 兩種常見 Disk：HHD (Hard Drive Disk, 硬碟) 與 SSD (Solid State Drive, 固態硬碟)



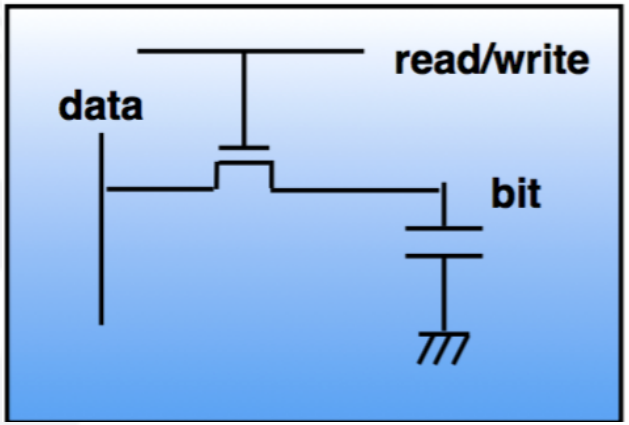
A Computer Schematic (5/5)

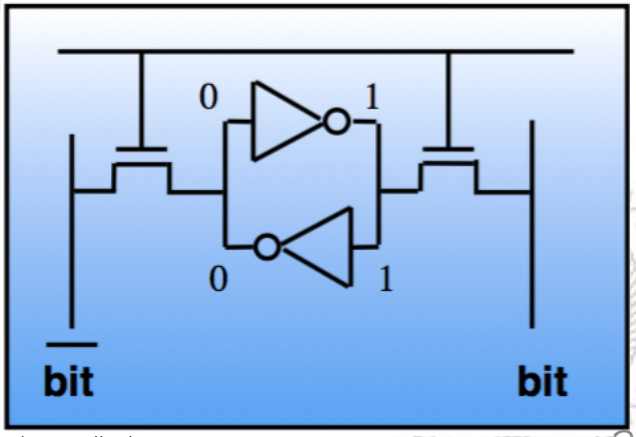
- 線的長度代表與 CPU 的相對距離大小
  - 愈長愈遠搬動資料的速度就愈慢
- Input/Output > Disk (100M Cycles) > RAM (100 Cycles) > Resgisters



RAM

- 兩種主流：DRAM, SRAM
- 佔一塊晶片的大部分 (60%)
- DRAM (Dynamic RAM)
  - 體積小
  - 便宜但慢: need refreshing
  - 主記憶體
  - DDR (Double Data Rate): Access in both rising/falling edge
- SRAM (Static RAM)
  - 體積大
  - 貴但快: no refreshing
  - Cache
  - Low power, costlier





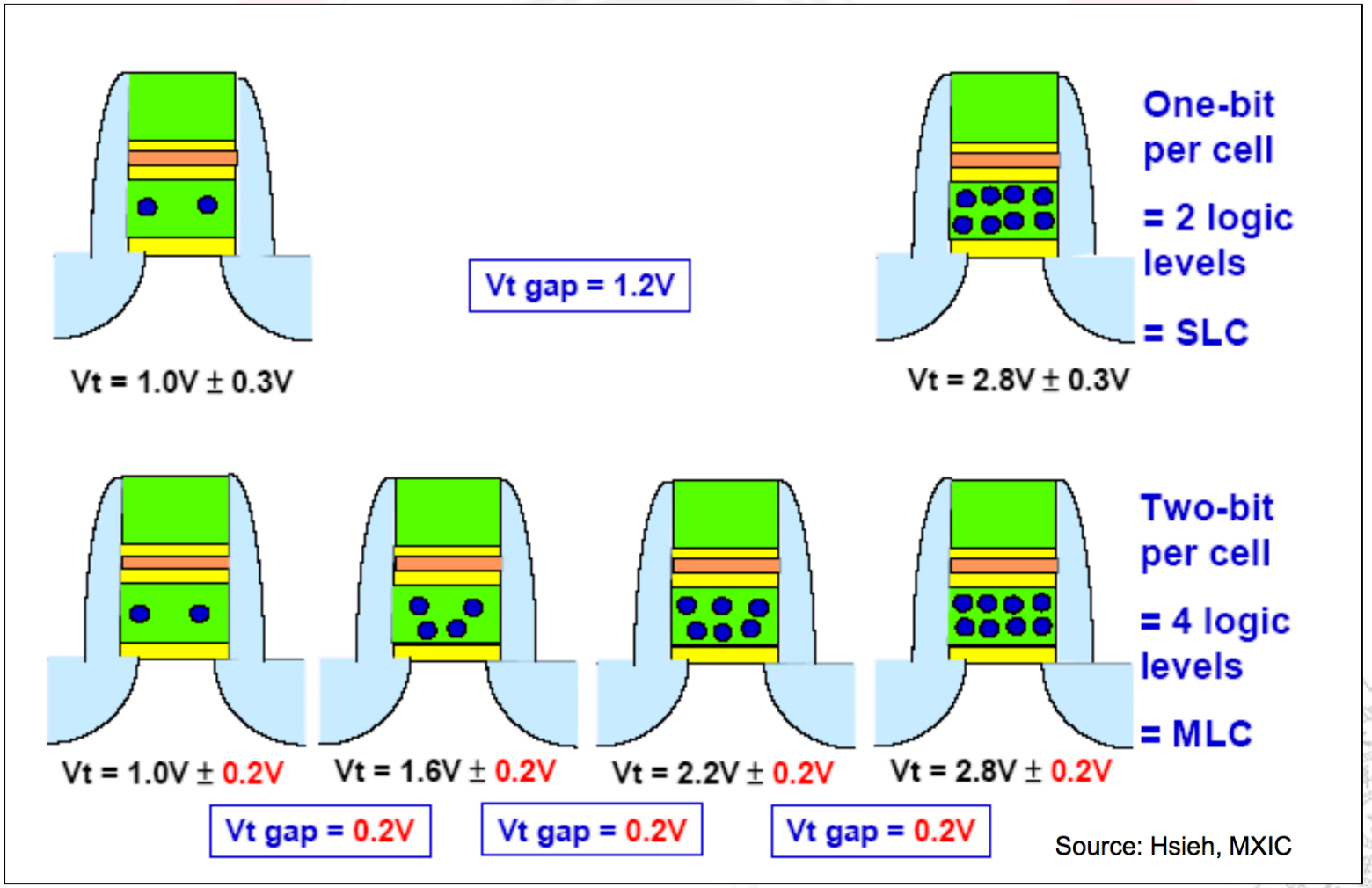
ROM

- 只可寫一次
- PROM: Programmable ROM (OTP, One-Time Programmable)
  - burn the fuses
- Mask ROM: Copy Protection
- 可重複讀寫
- EPROM: Erasable PROM
  - Electrically programmable
- UV-EPROM: Ultraviolet EPROM
  - Use UV-light to erase
- EEPROM: Electrically EPROM
  - Electrically programmable and erasable

ROM

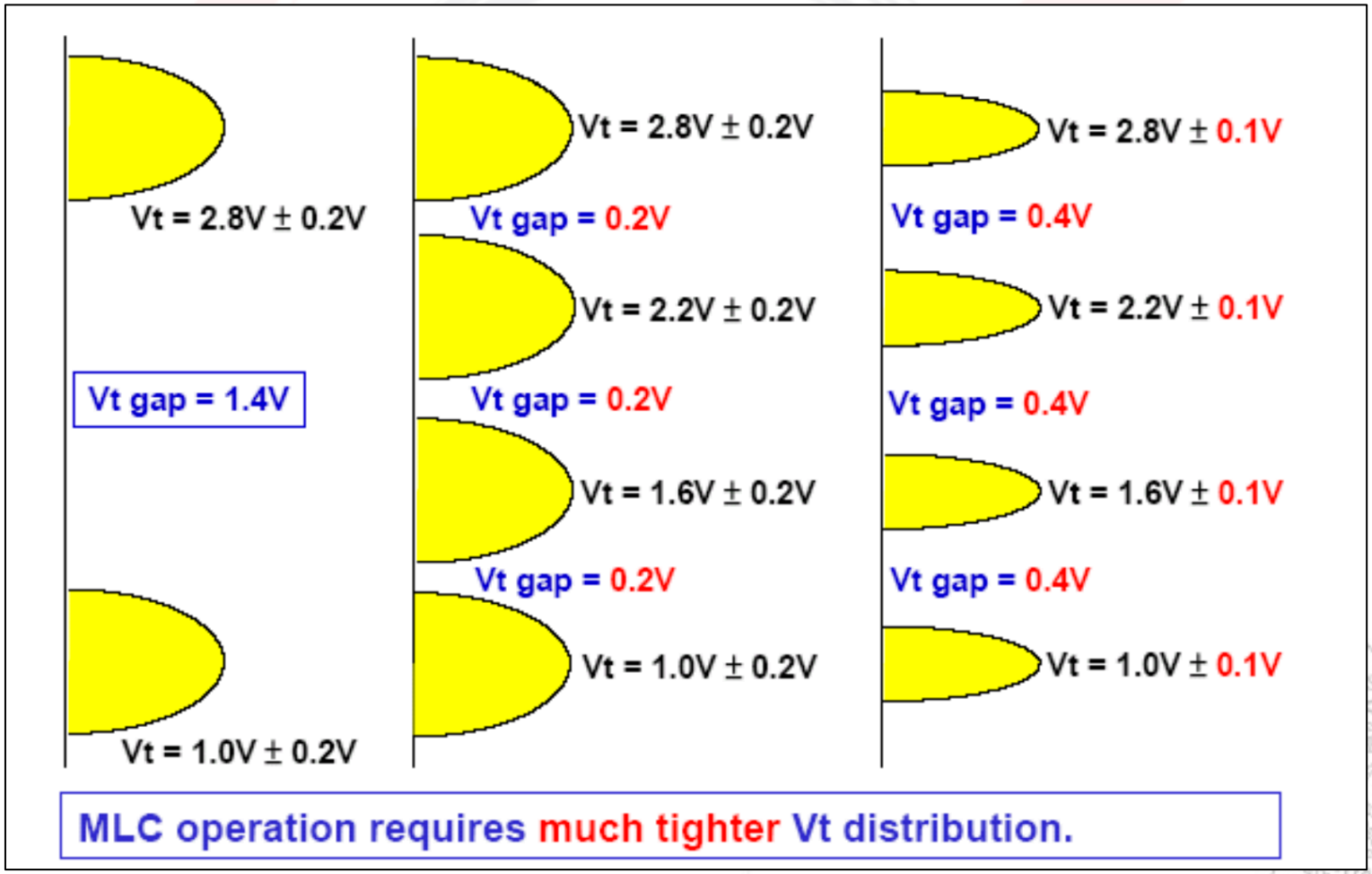
- Flash (快閃記憶體): 一種 EEPROM
  - SLC (single-level cell): 1-bit/cell, expensive, faster
  - MLC (multi-level cell): 多bits/cell, cheaper, slower
- NAND-flash
  - Cheaper, costlier
- NOR-flash

SLC vs. MLC (1/2)



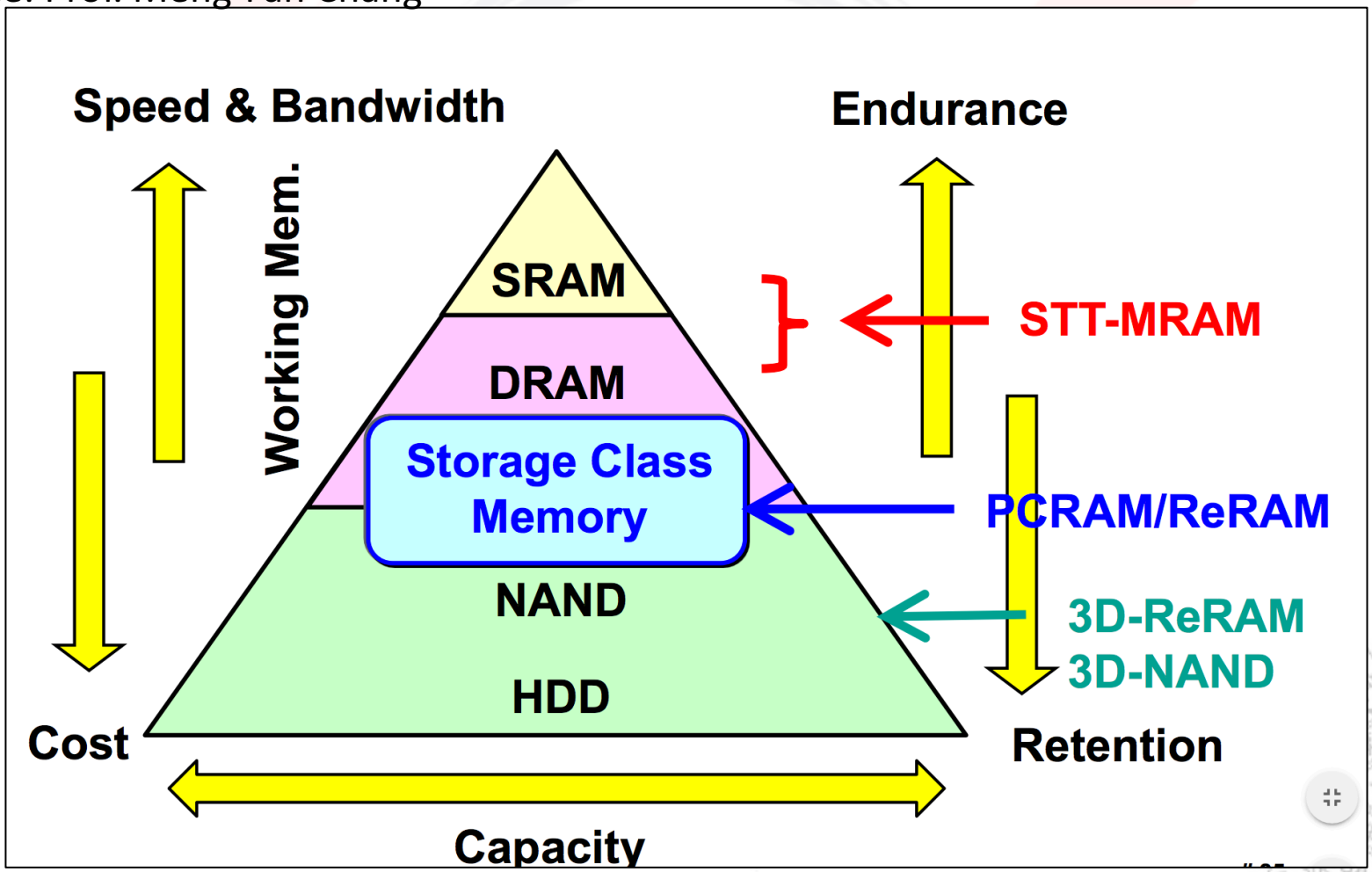
SLC vs. MLC (2/2)

Source: Prof. Meng-Fan Chang



Conclusion

Source: Prof. Meng-Fan Chang

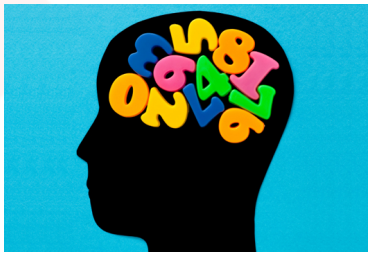


變數 (Variables)

Basic C++ Elements

- 變數 (Variables)
  - 有名稱，有型別，有記憶體位置。
  - 能見度 (scoping)：區域 (local) vs. 全域 (global)
  - 生存期間 (lifetime)
- 常數 (Literals)
  - 沒有名稱，有型別，有記憶體位置。
- 運算子 (Operators)
  - 預先定義好的對資料能做的「操作」，type-specific。如：加減法
  - 與變數或常數組合成「表示式」(expressions)。
  - 有優先次序 (precedence) 和順序關聯性 (associativity)。

Question: 人類與電腦怎麼記東西的？



- 大腦某些區域的神經元或神經元網路產生變化
- 越複雜的東西要花越久的時間去記憶

- 電腦的記憶體某些區域的位元或某些位元產生變化
- 越複雜的東西要花越大的空間去記憶

- 那個東西叫作「變數」(variables)
- 因此一個變數會有以下三種資訊(變數三要素)
  - 「資料型別」(data types)： 不同類型的資料要花費的記憶體空間大小不同。
  - 「變數名稱」(id)：用來稱呼該變數。
  - 「記憶體位置」(address)：記在記憶體的哪裡。

資料型別 (Data Types)

- `bool` 布林值 (Boolean) (只有 2 種可能(`true` or `false`) ，大小為 1 bit)
- `char` 字元 (character) (almost all 8 bits = 1 byte -- ASCII centric)
- 整數 (integer)：
  - `short` 短整數 (usually 16 bits = 2 bytes; could be same as int)
  - `int` 整數 (at least 16 bits, usually 32 bits = 4 bytes)
  - `long` 長整數 (usually 32 or 64 bits; could be same as int = 4 bytes)
  - `long long` 長長整數 (8 bytes)
- 浮點數 (floating point)：
  - `float` 單精確度浮點數 (usually 32 bits = 4 bytes, could be same as double)
  - `double` 雙精確度浮點數 (at least 32 bits; usually 64 bit = 8 bytes)
- 可以利用 `sizeof` 運算子得到變數或型別的大小。(也可以得變數大小)
  - 例： `cout << sizeof(int) << endl;`
- 可以記最多的型別: `long double` (16 bytes)

變數名稱 (Variable Names, Identifiers)

- "Identifier" (ID) = 使用者自己定義的名字
  - 在同個 scope 下不能重複
- 有效ID (Valid identifiers):
  - 以英文字母或底線(underscore (_))開頭
  - 後接任意數字、字母、底線
- 大小寫敏感 (case-sensitive)
  - "sum" 跟 "Sum" 不同名稱
- 不可以用數字開頭

Examples

- 合法 (Legal)
  - `i`
  - `wordsPerSecond`
  - `words_per_second`
  - `_green`
- 不合法 (Illegal)
  - `10sdigit`
  - `ten'sdigit`
  - `done?`
  - `double`

系統保留字 (Reserved Words, Keywords)

- Keywords: 變數名稱不可用這些字

```
alignas (since C++11)
alignof (since C++11)
and
and_eq
asm
atomic_cancel (TM TS)
atomic_commit (TM TS)
atomic_noexcept (TM TS)
auto(1)
bitand
bitor
bool
break
case
catch
char
char16_t (since C++11)
char32_t (since C++11)
class(1)
compl
concept (concepts TS)
const
constexpr (since C++11)
const_cast
continue
decltype (since C++11)
default(1)
delete(1)
do
double
dynamic_cast
else
enum
explicit
export(1)
extern(1)
false
float
for
friend
goto
if
inline(1)
int
import (modules TS)
long
module (modules TS)
mutable(1)
namespace
new
noexcept (since C++11)
not
not_eq
nullptr (since C++11)
operator
or
or_eq
private
protected
public
register(2)
reinterpret_cast
requires (concepts TS)
return
short
signed
sizeof(1)
static
static_assert (since C++11)
static_cast
struct(1)
switch
synchronized (TM TS)
template
this
thread_local (since C++11)
throw
true
try
typedef
typeid
typename
union
unsigned
using(1)
virtual
void
volatile
wchar_t
while
xor
xor_eq
```

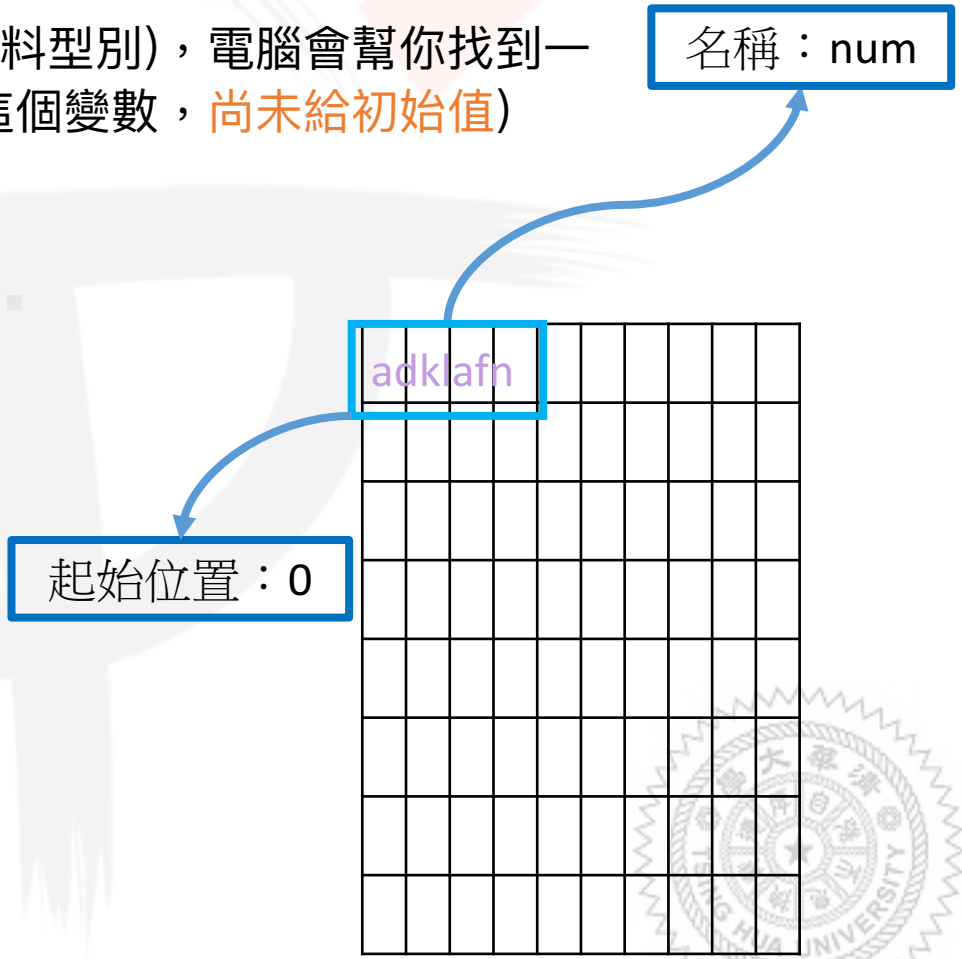
Source: http://en.cppreference.com/w/cpp/keyword

變數宣告 (Variable Declaration)

- 變數必須先宣告才可使用！

(先告訴電腦需要多大的記憶體空間(利用資料型別)，電腦會幫你找到一個這麼大的空間，並保留該空間給你存放這個變數，尚未給初始值)

- 型式： `type name;`
- Example:
  - `int num;`
  - `float pi;`
  - `char ch;`
  - `bool isDead;`
- 可同時宣告多個變數(以逗號分隔)
- Examples:
  - `int i, j, k;`
  - `float f, g, h;`



上次的例子

```
#include <iostream>
using namespace std;
int main(){
    int a;
    cout << "Please input a number: ";
    cin  >> a;
    cout << "The square of " << a << " is "
         << a*a << endl;
}
```

變數宣告

變數使用

變數使用

常數 (Literals) (1/2)

- 整數 (Integer)
  - `0`
  - `1`
  - `123` /* decimal */
  - `-123`
  - `0123` /* octal: $123_8 = 83_{10}$ */
  - `0x123` /* hexadecimal: $123_{16} = 291_{10}$ */
- 浮點數 (Floating point)
  - `6.023`
  - `6.023e23` /* 6.023×10^{23} */
  - `5E12` /* 5.0×10^{12} */

常數 (Literals) (2/2)

- 字元 (Character) – 用單引號括起 (single-quotes)
  - `'c'`
  - `'0'`
  - `'1'`
  - `'\n'` /* newline */
  - `'\t'` /* indent (tab) */
  - `‘\xA’` /* ASCII 10 (= 0xA) */
- Question: 單引號字元、反斜線字元如何表示？
  (提示：利用跳脫字元 (escape charater): `'\'`)
- Question: `'0'` 跟 `0` 差在哪？

More about 字元常數
ASCII Characters

- American Standard Code for Information Interchange, 美國資訊交換標準代碼
- wiki: https://zh.wikipedia.org/wiki/ASCII
- ASCII: Maps 128 characters to 7–bit code.
  - 兩種： printable ($\geq 20_{16}(= 32_{10})$) and non–printable (ESC, DEL, etc $< 20_{16}(= 32_{10})$) characters

ASCII Table

- 重要的
- ‘A’ = 65
- ‘a’ = 97
- ‘ ‘ = 32
- ‘\0’ = 0
- ‘0’ = 48

Table E.2 The Standard ASCII Table

ASCII Character	Dec	Hex	ASCII Character	Dec	Hex	ASCII Character	Dec	Hex	ASCII Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	,	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F

0 vs '0'

- 數字 `0` 的值是 `0`
- 字元 `'0'` 的值是 `48`

- 字元 `' '` 的值是 `32`
- 字元 `'A'` 的值是 `65`
- 字元 `'a'` 的值是 `97`

- 想一想 1：如何讓字元的 `'0','1','2'…` 變成 `0,1,2…`？
  - 答： `- '0'`
- 想一想 2：如何讓大寫的英文字母轉成小寫的英文字母？
  - 答： `+ ('a' - 'A')`

More on 整數常數 (Integer Literals)

- 十進位 (Decimal)
  - `123 /* positive */`
  - `-123 /* negative */`
- 十六進位 (Hexadecimal): `0x` 開頭
  - `0x123 /* upper bits are 0 = 0x00000123 */`
  - `0xfeedface /* full 32 bits */`
- 八進位 (Octal): `0` 開頭
  - `012 /* octal 8 + 2 = 10 */`
  - `098 /* illegal! Digits must be 0-7 */`

跳脫字元(Escape Character)與跳脫序列(Escape Sequence)

- 跳脫字元（'\'）讓後接的字元「有特殊功能變無特殊功能」或「無特殊功能變有特殊功能」，整體叫作跳脫序列

表 3.2.2 常用的跳脫序列

跳脫序列	所代表的意義	ASCII 十進位值	ASCII 十六進位值
\a	警告音(Alert)	7	0x7
\b	倒退一格(Backspace)	8	0x8
\n	換行(New line)	10	0xA
\r	歸位(Carriage return)	13	0xD
\t	跳格(Tab)	9	0x9
\0	字串結束字元(Null character)	0	0x0
\\	反斜線(Backslash)	92	0x5C
\'	單引號(Single quote)	39	0x27
\"	雙引號(Double quote)	34	0x22

字串常數(String Literals)

- 字串 = 一串字元連起來
  - 由兩個雙引號括起 (double quotes (""))
  - 資料型別是字元陣列 (char array)
  - C++ 裡還有另一種代表字串的資料型別是字串 (string)
- Example
  - `"This is a text string consisting of a sequence of chars"`
  - `"To include double-quote in a string, do \"this\" kind of escape"`
- 字串 (char array) in C++ are null–terminated
  - 在字串結尾有一個隱藏的空字元 `'\0'` (value == 0)
  - 如果在字串中放 `\0`, 那麼字串就會被終止！
    - you may get a warning from the compiler
- 以下這行會印出什麼？
  - `cout << "hello\0world\n";`

上次的例子

```
#include <iostream>
#define CSW 18
using namespace std;
int main(){
    int a;
    cout << "Please input a number: ";
    cin  >> a;
    cout << "The square of " << a << " is "
         << a*a << endl;
    cout << "By the way, 1 + 1 = " << 2 << endl;
}
```

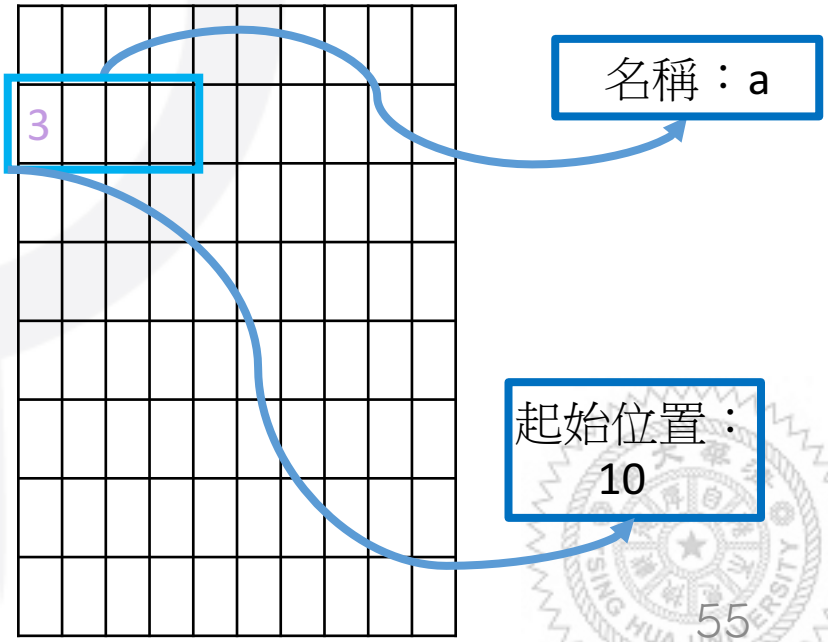
白框中都是常數

變數的宣告(Declaration)與初始化 (Initialization)

- 變數可在宣告的同時並初始化 (給初始值)
- 型式：
```
type name = value;
```
- Example:
  - `int age = 18;`
  - `float pi = 3.1415;`
  - `char q = 'Q';`
  - `bool isPassed = false;`
- 可同時宣告與選擇初始化多個變數(以逗號分隔)
- Examples:
  - `int i = 3, j, k = 5; // j is not initialized`
  - `float f, g = 6.789, h = -0.4689;`
    - `// f is not initialized`

指派運算子 (Assignment Operator (=))

- = 不是等於(equal)的意思，而是得到(get)的意思。
- `int a = 3;`
- 電腦先依型別是 `int`，得知需要 4 bytes 的空間，在記憶體中找到一個沒人用的 4 bytes 的空間(房間)，命名叫作 a，並把 3 這個值丟進去(指派，assign)。
- 由 a (房間)來看，它得到(get)了 3。



- 所以

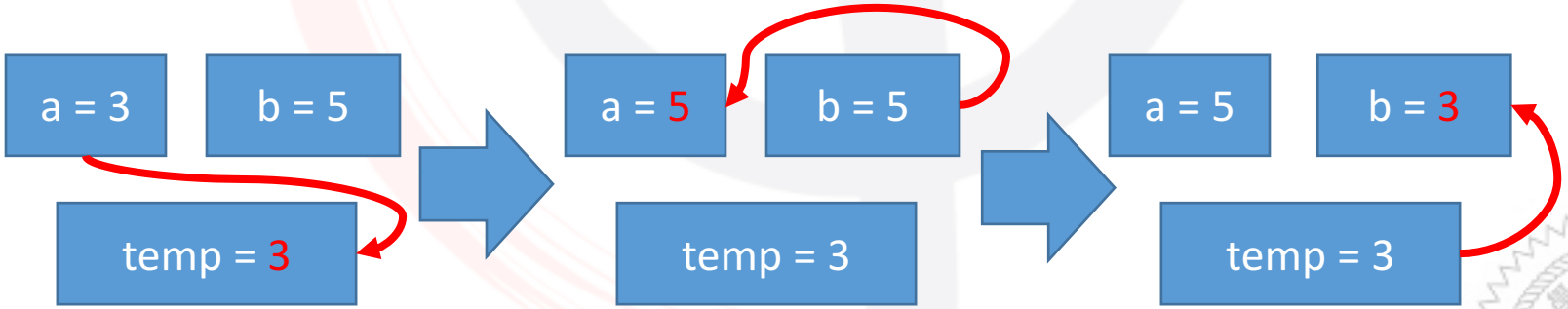
```
int a = 3, b = 5;
a = b;
b = a;
```

- a 是多少？ b 是多少？
- 如何交換 a, b 的值？

如何交換(swap) a, b 的值？

- The 1st Algorithm: Swap Algorithm
- 定義一個新變數 temp

```
int a = 3, b = 5, temp;
temp = a;
a = b;
b = temp;
```



變數的能見度(Scope)與生存時間(Lifetime)

- Scope: 從宣告那行到遇到自己那層的右大括號為止。
- Lifetime: 看記憶體位置而定
  - 堆積 (heap):
    - 程式開始執行到程式終止 (global variables, static local, …)
    - new 到 delete (動態) (第 6 章)
  - 堆疊 (stack):
    - 從宣告那行到遇到自己那層的右大括號為止。(local variables, parameters, …)

local_var 的能見度與生存時間

global_var 的能見度與生存時間

local_var 的能見度與生存時間

abc.cpp

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

int global_var = 7; // 全域變數

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

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

```

a 的能見度與生存時間

local_var 的能見度與生存時間

global_var 的能見度與生存時間

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

int global_var = 7; // 全域變數 (global variable)

int main(int argc, char **argv)
{
    int local_var = 3; // 區域變數 (local variable)

    {
        int a = 5; // 區域變數 (local variable)
        cout << "\tglobal_var = " << global_var << endl;
        cout << "\tlocal_car = " << local_var << endl;
        cout << "\ta = " << a << endl;
    }
    cout << "a = " << a << endl; // Compile error, 找不到 a
    cout << "global_var = " << global_var << endl;
    cout << "local_car = " << local_var << endl;

    return 0;
}
```

When do we use variables?

- 當你想要先讓電腦記著「某個東西」(不一定是數字)，等一下會再用到時。
- 例：使用者的輸入、小精靈的剩餘生命數、小精靈是否死亡(死亡狀態)、第幾關(關卡數)、張舜為每天的睡覺時數、張舜為每日一句、GOTC成立年數……。
- `int remain_lives = 3;`
- `bool isDead = false; // flag`
- `int level = 1;`
- `double GOTC_years = 1.4;`
- `int CSW_sleep_hours_per_day = 3;`
- `char GG3vs0_classic_quotes[] = "太神啦！";`
- `string GG3vs0_saying = "太猛啦！";`

變數 vs. #define

- `double PI = 3.1415;`
- 變數
- 變數三要素：名稱、型別 (type) 和記憶體位置
- 可以在程式碼中改值(runtime 時改值)： `PI = 3.14;`

- `#define PI 3.1415`
- 常數，沒有記憶體位置
- preprocessor 做 macro substitution 處理
- 不可以在程式碼中改值： ~~`PI = 3.14;`~~

const vs. #define

- `const double PI = 3.1415;`
- 常數 (但不是指 literals)，而是代表不能再被改值的變數
- 變數三要素：名稱、型別 (type) 和記憶體位置
- 不可以在程式碼中改值(compile error)： ~~`PI = 3.14;`~~

- `#define PI 3.1415`
- 常數，沒有記憶體位置
- preprocessor 做 macro substitution 處理
- 不可以在程式碼中改值： ~~`PI = 3.14;`~~

Outline

- Bits and Bytes
- Variables
- Operators
- Some Practices

Operators

運算子 (Operators)

- 用來操作變數，變數類型不同操作會不同。
  - 數字 + 數字 vs 字串 + 字串 vs ~~數字 + 字串~~
- 變數與運算子組成表達式(expressions)與陳述式(statements)

表達式 (Expression)

- Definition: An ***expression*** is any combination of variables, constants, operators, and function calls.
- Every expression has a value and a type, which derived from the types of its components.
- Examples:
  - `a = 10` // type 為 a 的 type，值為 a
  - `a` // type 為 a 的 type，值為 a
  - `a * a` // type 為 a 的 type，值為 a^2
  - `a == 10` // type 為 bool，值為 true or false
  - `a < 10` // type 為 bool，值為 true or false
  - `x + sqrt(y)` // type 為 double，值為 $x + \sqrt{y}$
  - `foo(y)` // type 為 foo() 的 return type
  - `x & z + 3 || 9 - w-- % 6`

陳述式 (Statement) (1/2)

- Definition: A ***statement*** expresses a complete unit of work.
- Executed in sequential order
- 簡單陳述式 (Simple statement): 分號 (semicolon) 結尾
  - `z = x * y + 5; // NOT z = xy + 5;`
  - `y = y + 1;`
  - `y++;  // same as y = y + 1;`
  - `y += 1;  // same as y = y + 1;`
  - `x = abs(y);  // 把回傳值存在變數 x 裡`
  - `abs(y);  // 不把回傳值存起來`
  - `;  // null statement`
- 通常一行一個 simple statement

陳述式 (Statement) (2/2)

- 組合陳述式 (Compound statement): groups simple statements using braces (大括號).
- Equivalent to a simple statement syntactically
  - `{ z = x * y;  y = y + 1; } // bad coding style`
  - do this:

```
{
        z = x * y;
        y = y + 1;
}
```

- 通常一行一個 simple statement

Operators

- 運算子三要素：功能、優先次序和順序關聯性
- 功能 (Function): What does it do? What data type(s) does it operate on? What is the resulting type?
- 優先次序 (Precedence): in which order are operators combined?
  - Example:
    - "a * b + c * d" is the same as "(a * b) + (c * d)"
    - 先乘除、後加減 (加減法的 precedence < 乘除的 precedence)
- 順序關聯性 (Associativity): in which order are operators of the same precedence combined?
  - Example:
    - "a – b – c" is the same as "(a – b) – c"
    - 因為加/減法是從左到右 (associate left-to-right)
- http://en.cppreference.com/w/cpp/language/operator_precedence

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right

2	`++ --`
	`()`
	`[]`
	`.`
	`->`
		Suffix/postfix increment and decrement
		Function call
		Array subscripting
		Element selection by reference
		Element selection through pointer	

3	`++ --`
	`+ -`
	`! ~`
	`(`*type*`)`
	`*`
	`&`
	`sizeof`
	`new, new[]`
	`delete, delete[]`
		Prefix increment and decrement
		Unary plus and minus
		Logical NOT and bitwise NOT
		Type cast
		Indirection (dereference)
		Address-of
		Size-of
		Dynamic memory allocation
		Dynamic memory deallocation
			Right-to-left

4	`.* ->*`	Pointer to member	Left-to-right
5	`* / %`	Multiplication, division, and remainder	
6	`+ -`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=` `> >=`	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
9	`== !=`	For relational = and ≠ respectively	
10	`&`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	

15	`?:`
	`=`
	`+= -=`
	`*= /= %=`
	`<<= >>=`
	`&= ^= |=`
		Ternary conditional
		Direct assignment (provided by default for C++ classes)
		Assignment by sum and difference
		Assignment by product, quotient, and remainder
		Assignment by bitwise left shift and right shift
		Assignment by bitwise AND, XOR, and OR
			Right-to-left

16	`throw`	Throw operator (for exceptions)	
17	`,`	Comma	Left-to-right

指派運算子 = (Assignment Operator) (1/3)

- Precedence: 15 (1最高、17最低)
- Associativity: right to left (a = b = c <=> a = (b = c))
- Function: Changes the value of a variable.
  - `x = x + 4;`

- The value on the right-hand-side (RHS) must be convertible to the variable's type on the left-hand-side (LHS)
- Assigning float to int variable => truncation (unboxing, downcasting)
  - `int pi = 3.1415; // actually pi gets 3`
- Assigning int to float variable => int-to-float conversion (boxing, auto-boxing, upcasting)
  - `float pi = 3;    // actually pi gets 3.0`

指派運算子 = (Assignment Operator) (2/3)

- Recall: 任何 expression 都有 value 與 type
  - `a = 10  // type 為 a 的 type，值為 a`
- Recall: Assignment associates right to left.
- Thus:
  - `y = x = 3;`  `y = (x = 3);`
  - y gets the value 3, because (x = 3) evaluates to the value 3.

指派運算子 = (Assignment Operator) (3/3)

- without assignment vs with assignment
  - `x + 1;` vs `x = x + 1;`

```
1  #include <iostream>
2  using namespace std;
3
4  int main(int argc, char **argv){
5      int x = 3;
6      x + 1;  // 沒有將加完後的結果塞回給 x => x 不變
7      cout << "Without assignment: x = " << x << endl;
8      x = x + 1; // 將加完後的結果塞回給 x => x 得到 x + 1
9      cout << "With assignment: x = " << x << endl;
10     return 0;
11 }
12
```

- `Without assignment: x became 3`
- `With assignment: x became 4`

Expressions vs. Statements

- 在 C++ 裡，一個表達式加分號就是一個陳述式
- But an expression can be useless if it does not have "side effects"
  - Side effects: assignment, input/output, etc.
  - It can evaluate a lot of things, then throw away the results!
  - `x + 1;  // useless`
  - `x = x + 1;  // not useless`

算術運算子 (Arithmetic Operators)

Symbol	Operation	Usage	Precedence	Assoc
*	multiply	x * y	6	l-to-r
/	divide	x / y	6	l-to-r
%	modulo	x % y	6	l-to-r
+	addition	x + y	7	l-to-r
-	subtraction	x - y	7	l-to-r

- Associativity: left to right
- % 的功能是取餘數
- *, /, % have higher precedence than +, -.

Type Conversion in Arithmetic Expressions (1/2)

- Type Conversion = Type Casting
- 混合 type 的運算，小 type 會晉升 (promote) 到大 type
  - `x + 4.3 // if x is int, converted to double and result is double`
- 整數除整數，仍然是整數 (fraction is dropped)
  - `x / 3 // if x is integer 5, quotient is 1 (not 1.666666...)`
  - How to fix it?

```
1. x / 3.0
2. (double) x / 3; // explicitly type casting
```

Type Conversion in Arithmetic Expressions (2/2)

- 取餘數 (Modulo) -- result is remainder.
  - `x % 3 // if x is int and x is 5, result is 2.`
  - But if x is int and x is -5, result is -2.
  - Recall: In abstract algebra, $-2 \equiv 1 \, in \, \mathbb{Z}_3$
  - How to achieve this finite group's property? (homework)
- For assignment, type and value of expression is converted to match the RHS

```
double x, z;
int y;
x = y = z = 3.14;
```

  - Result: z = 3.14; y = 3; x = 3.0;
  - so, x != z

位元運算子 (Bitwise Operators)

Symbol	Operation	Usage	Precedence	Assoc
~	bitwise NOT	~x	4	r-to-l
<<	left shift	x << y	8	l-to-r
>>	right shift	x >> y	8	l-to-r
&	bitwise AND	x & y	11	l-to-r
^	bitwise XOR	x ^ y	12	l-to-r
|	bitwise OR	x | y	13	l-to-r

- Refer to page 10 ~ 11
- Operate on variables bit–by–bit.
- $x = 10010100_2$, x >> 4, $x = 11111001_2$
- http://acm.nudt.edu.cn/~twcourse/BitwiseOperation.html

Examples

- `x >> a; // 往右移 a bits`
- `x = 10100100`$_2$`, x = x >> 4 // x is 11111010`$_2$
- `x = x & 00001111; // x is 00001010` (masking)

- `y = 0x12CF34F9 =`
  `0001|0010|1100|1111|0011|0100|1111|1001`
- 取出 F9? `y = y & 0x000000FF; // = 0xFF`
- 取出 34? `y = (y >> 8) & 0xFF;`
- 得到 0xFFFF34F9? `y = y | 0xFFFF0000;`

關係運算子 (Relational Operators)

Symbol	Operation	Usage	Precedence	Assoc
>	greater than	x > y	9	l-to-r
>=	greater than or equal	x >= y	9	l-to-r
<	less than	x < y	9	l-to-r
<=	less than or equal	x <= y	9	l-to-r
==	equal	x == y	10	l-to-r
!=	not equal	x != y	10	l-to-r

- Result is 1 (true) or 0 (false)
- 注意：不要搞混 equality (==) 與 assignment (=)
  - x == y vs. x = y

邏輯運算子 (Logical Operators) (1/3)

Symbol	Operation	Usage	Precedence	Assoc
!	logical NOT	!x	4	r-to-l
&&	logical AND	x && y	14	l-to-r
||	logical OR	x || y	15	l-to-r

- Result is 1 (true) or 0 (false)
- 注意：不要搞混 bitwise operators 與 logical operators
  - ~x vs !x
  - x & y vs x && y
  - x | y vs x || y

邏輯運算子 (Logical Operators) (2/3)

- Some examples

1.如果是老人或小孩一律免費

- if age >= 60 or age < 12, then free
- ```
  if ((age >= 60) || (age < 12)) { /* free */ }
  else { /* Pay the money */ }
  ```

2.如果死了，Game Over

- ```
  if (isDead == true) { /* Game over*/ }
  ```

3.高富帥(富or高and帥)，且不抽菸、喝酒、吸毒

- if money > 1 billion, or height >= 183 and isHandsome, no cigarette, no alcohol, and no drugs, then I marry him.
- ```
  if ((money > 1000000000 /* USD */ || (height >= 183 &&
  isHandsome)) && !cigarette && !alcohol && !drugs)
  { /* I marry him */ }
  else { /* No way */ }
  ```


  ```

邏輯運算子 (Logical Operators) (3/3)

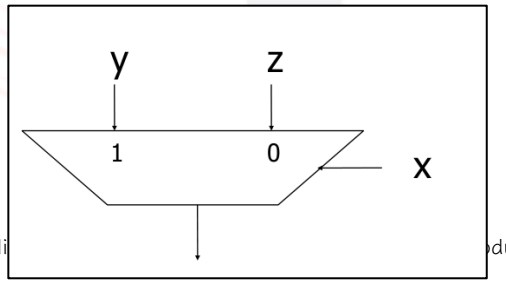
- 注意，在 C++ 語言裡
  - 0 為 false
  - 非 0 為 true
- 所以
- `int a = 5;`
  - a is not 0 (true)
  - !a is 0 (false)
- `int b = 0;`
  - b is 0 (false)
  - !b is 1 (true)

條件運算子 (Conditional Operator) (1/2)

- 又稱三元運算子(ternary operator)
- like if, else…

Symbol	Operation	Usage	Precedence	Assoc
?:	conditional	x?y:z	16	l-to-r

- If x is true (non-zero), result is y;
- else, result is z.
- Like a MUX, with x as the select signal.



條件運算子 (Conditional Operator) (2/2)

- Some examples

```
1.isDead = (hp <= 0) ? true : false;
2.(hp <= 0) ? (isDead = true) : (isDead = false);
3.cout << (isDead ? "Dead!" : "Alive!") << endl;

4.isEnemyNearby ? kill() : wait();

5.x = (x > 0) ? x : -x;     // x = abs(x);
```

遞增/遞減運算子 (Increment/Decrement Operator)

- Changes value of variable before (or after) its value is used in an expression.

Symbol	Operation	Usage	Precedence	Assoc
++	postincrement	x++	2	r-to-l
--	postdecrement	x--	2	r-to-l
++	preincrement	++x	3	r-to-l
--	predecrement	--x	3	r-to-l

- Pre: Increment/decrement variable before using its value.
- Post: Increment/decrement variable after using its value.

Using ++ and -- (1/3)

- x++：先用再加加
- ++x：先加加再用

Using ++ and -- (2/3)

```
x = 4;
y = x++;
```

=

```
x = 4;
y = x;
x = x + 1;
```

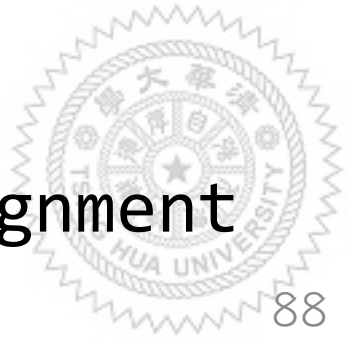
- Results: x = 5, y = 4
- Because x is incremented after assignment

```
x = 4;
y = ++x;
```

=

```
x = 4;
x = x + 1;
y = x;
```

- Results: x = 5, y = 5
- Because x is incremented before assignment



Using ++ and -- (3/3)

```
x = 4;
y = x++ + 3;
```

=

```
x = 4;
y = x + 3;
x = x + 1;
```

- Results: x = 5, y = 7
- Because x is incremented after assignment

```
x = 4;
y = ++x + 3;
```

=

```
x = 4;
x = x + 1;
y = x + 3;
```

- Results: x = 5, y = 8
- Because x is incremented before assignment

Practice with Precedence

- Assume a = 3, b = 2, c = 3, d = 4.
- x = a % b + c * d / 2; /* x = 7 */
- same as:
- x = (a % b) + ((c * d) / 2);
- 若怕搞混順序，請善用括號 ()，括號內的表達式會先被執行。(因為你也背不起來 precedence table)
- 原則：
- 算術運算子 > 關係運算子 > 邏輯運算子 > 指派運算子
  - e.g. 3 < x + y < 15
  - x + y > 3 && x + y < 15
  - (((x + y) > 3) && ((x + y) < 15))

特殊指派運算子 (Special Assignment Operators) (1/2)

- arithmetic assignment or bitwise assignment

Statement	Equivalent assignment
`x += y;`	`x = x + y;`
`x -= y;`	`x = x - y;`
`x *= y;`	`x = x * y;`
`x /= y;`	`x = x / y;`
`x %= y;`	`x = x % y;`
`x &= y;`	`x = x & y;`
`x |= y;`	`x = x | y;`
`x ^= y;`	`x = x ^ y;`
`x <<= y;`	`x = x << y;`
`x >>= y;`	`x = x >> y;`

All have same precedence and associativity as = and associate right-to-left.

特殊指派運算子 (Special Assignment Operators) (2/2)

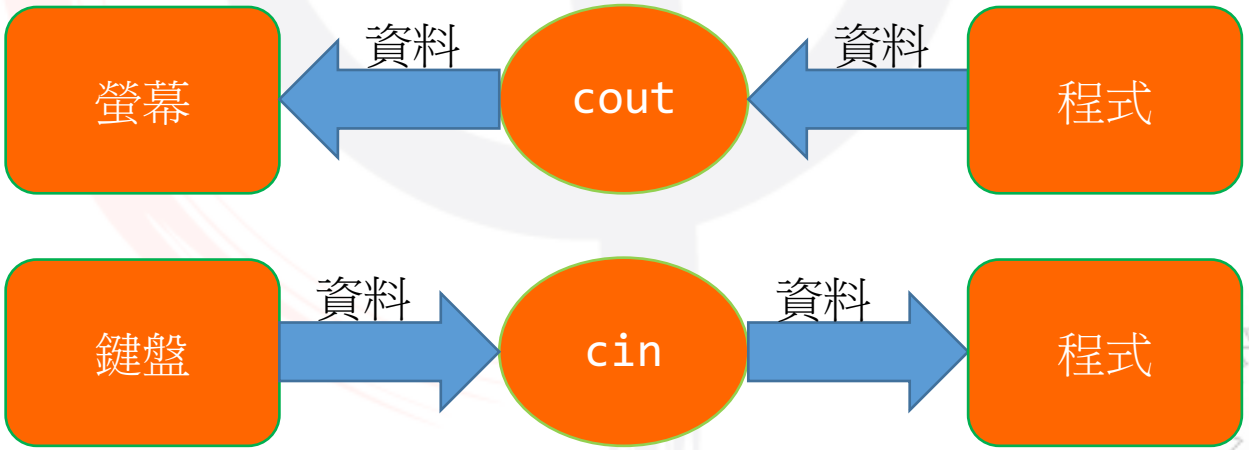
- `x *= x;`
- `x = x*x;`

- `x *= x + 1;`
- `x *= (x + 1);`
- `x = x*(x + 1);`

- `x /= 2;`
- `x = x/2;`

標準輸入 Standard Input (std::cin) (1/3)

- Recall cout
- cout 你可以想像是個程式的對外窗口，裡面是程式，外面是螢幕。
- cin 你可以想像是個電腦的對內窗口，裡面是程式，外面是鍵盤輸入。



Recall: A Program is like a Fuction

- 輸入 (input) –> 處理 (process) –> 輸出 (output)
- 輸入要用到 `cin` 將資料存進變數
- 輸出要用到 `cout` 將結果顯示於螢幕上
- 處理就是操作讀進來的資料(變數)，運用各種條件判斷、迴圈等技巧，但幾乎必用到所有學過的運算子

標準輸入 Standard Input (std::cin) (2/3)

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

int main(int argc, char **argv){
    int year, month, day;
    cin >> month >> day >> year;
    cout << "民國 " << year - 1911 << " 年 "
         << month << " 月 " << day << " 日" << endl;
    return 0;
}
```

標準輸入 Standard Input (std::cin) (3/3)

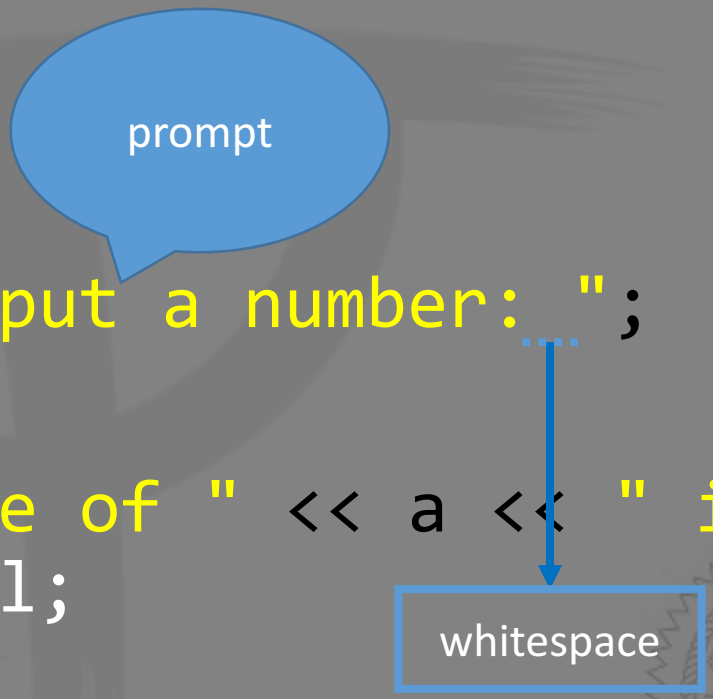
```
7 28 2015
民國 104 年 7 月 28 日

--------------------------------

Process exited after 6.289 seconds with return value 0
請按任意鍵繼續 . . .
```

How do users know what to type?

- 使用者要怎麼知道他應該鍵盤要輸入什麼東西呢？
- 利用提示訊息 (prompt)
- 先 cout 一段話告訴使用者要打什麼，通常後接一個空白，且不要換行字元。

上次的例子

```
#include <iostream>
using namespace std;
int main(){
    int a;
    cout << "Please input a number: ";
    cin  >> a;
    cout << "The square of " << a << " is "
         << a*a << endl;
}
```



Result

1. Prompt (ask user to input)

```
Please input a number:
```

2. User input

```
Please input a number: 7
```

3. Result

```
Please input a number: 7
The square of 7 is 49
```

Outline

- Bits and Bytes
- Variables
- Operators
- Some Practices

Some Practices

作業 Part I

- 撰寫一個程式，檔名叫作 JudgeLeap.cpp
  1. Prompt "enter the year (0~3999): "
  2. 使用者輸入年分
  3. 若該年為閏年， cout << year << " is a leap year.\n" ;
  4. 否則，cout << year << " isn't a leap year.\n";
- 閏年就是該年「可被 400 整除 或 可被 4 整除且不被 100 整除」
  - 1992、1996 are leap
  - 1600、2000、2400 are leap
  - 1800、1900、2100 are not leap
- You may use %
- You may use the ? : conditional operator.

Simple Results

```
enter the year (0~3999): 1982
1982 isn't a leap year.
```

```
enter the year (0~3999): 2000
2000 is a leap year.
```

```
enter the year (0~3999): 1984
1984 is a leap year.
```

```
enter the year (0~3999): 2001
2001 isn't a leap year.
```

```
enter the year (0~3999): 1900
1900 isn't a leap year.
```

```
enter the year (0~3999): 2004
2004 is a leap year.
```

作業 Part II

- 撰寫一個程式，檔名叫作 4digit.cpp
  1. Prompt "enter a number (0~9999): "
  2. 使用者輸入一個大小為 0~9999 的數字
  3. 將該數字拆成 4 個位數分別印出
  4. e.g.
     1) input: 4567, output: 4 5 6 7
     2) input: 321, output: 0 3 2 1
- 想一想：如何分別取出每一位數？
- You may use /, %

Simple Results

```
enter a number (0~9999): 0
0 0 0 0
```

```
enter a number (0~9999): 5050
5 0 5 0
```

```
enter a number (0~9999): 2
0 0 0 2
```

```
enter a number (0~9999): 5487
5 4 8 7
```

```
enter a number (0~9999): 204
0 2 0 4
```

```
enter a number (0~9999): 9000
9 0 0 0
```

```
enter a number (0~9999): 800
0 8 0 0
```

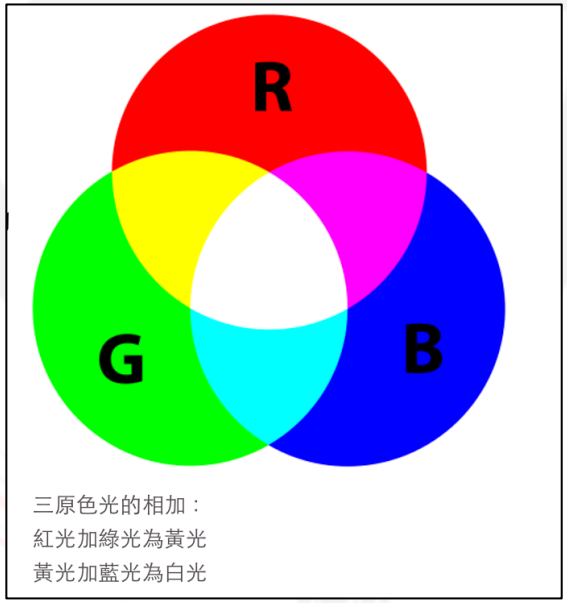
```
enter a number (0~9999): 9999
9 9 9 9
```

作業 Part III

- 製作一個色碼轉換器
- 先了解什麼是色碼
- 電腦如何儲存顏色資訊？
  - 其中一種方式：用 RGB (或RGBA)
  - A is alpha: 代表透明度

What is RGB? (1/2)

- **三原色光模式（RGB color model），又稱RGB顏色模型或紅綠藍顏色模型**，是一種加色模型，將紅（Red）、綠（Green）、藍（Blue）三原色的色光以不同的比例相加，以產生多種多樣的色光。



三原色光的相加：
紅光加綠光為黃光
黃光加藍光為白光

- 簡單來說，就是一種在電腦裡的顏色表示法！

What is RGB? (2/2)

- 一個顏色顯示的描述是由三個數值控制的，分別為R、G、B。但三個數值位為最大時，顯示為白色，當三個數值最小時，顯示為黑色。
- 常見數值表示方式如右

方式	RGB 表示
浮點	(1.0, 0.0, 0.0)
百分比	(100%, 0%, 0%)
八位數字	(255, 0, 0) 或 #FF0000 (十六進位)
十六位數字	(65535, 0, 0)

- 以 0 ~ 255 表示顏色的明度 (小 ~ 大)
  - 0xFF0000 => R: 255 (Max), G: 0 (Min), B: 0 (Min) => Red
  - 0xFFFFFF => R: 255 (Max), G: 255 (Max), B: 255 (Max) => White
  - 0x000000 => R: 0 (Min), G: 0 (Min), B: 0 (Min) => Black
  - 0x808080 => R: 128, G: 128, B: 128 => Gray
  - 0x0000FF => R: 0 (Min), G: 0 (Min), B: 255 (Max) => Blue
- 想要自己配色？ Click Here: Html Color Codes (http://html-color-codes.info/)

作業 Part III

- 製作一個 RGB to Color 色碼轉換器
- 檔名： rgb2color.cpp
- 輸入一個 rgb (e.g. 0xFFFFFF)，輸出每個顏色的十進位大小 (0~255)，以及他的深淺百分比 (0~100%)
  1. Prompt “Please type the rgb: ”
  2. `cin >> hex >> rgb; // how to cin hex format?`
     - http://stackoverflow.com/questions/13196589/getting-hex-through-cin
  3. `cout: Red = ? (?%), Green = ? (?%), Blue = ? (?%)`
- 問題：如何從一個數字中取出我要的片段？
- You may use the masking skill. (<<, >>, &, |, …)

Simple Results

```
[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0xffffff
Red = 255 (100%), Green = 255 (100%), Blue = 255 (100%)

[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0xff0000
Red = 255 (100%), Green = 0 (0%), Blue = 0 (0%)

[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0xffff00
Red = 255 (100%), Green = 255 (100%), Blue = 0 (0%)

[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0x0000ff
Red = 0 (0%), Green = 0 (0%), Blue = 255 (100%)

[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0x888888
Red = 136 (53.3333%), Green = 136 (53.3333%), Blue = 136 (53.3333%)

[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0x808080
Red = 128 (50.1961%), Green = 128 (50.1961%), Blue = 128 (50.1961%)

[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0x123456
Red = 18 (7.05882%), Green = 52 (20.3922%), Blue = 86 (33.7255%)

[Slighten@SlightenCheng ttys000-bash 07:45 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0
Red = 0 (0%), Green = 0 (0%), Blue = 0 (0%)

[Slighten@SlightenCheng ttys000-bash 07:46 ~/codes/c++/hw2 $ ./a.out
Please type the rgb: 0x00000f
Red = 0 (0%), Green = 0 (0%), Blue = 15 (5.88235%)
```

作業 Part IV

- 製作一個 Color to RGB 色碼轉換器
- 檔名： color2rgb.cpp
- 輸入Red, Green, Blue (0~255)，輸出 RGB

1. Prompt “Red = ”; `cin >> red;`
2. Prompt “Green = ”; `cin >> green;`
3. Prompt “Blue = ”; `cin >> blur;`
4. `cout: RGB is #FFFFFF (e.g.)`

- 問題：如何輸出 uppercase hex (http://www.cplusplus.com/reference/ios/uppercase/)，如何確保 0 印出 00, 1 印出 01, …(cout 2 digits: http://www.cplusplus.com/forum/beginner/13082/)
  - cout << ? << ? << … << red;
  - cout << ? << ? << … << green;
  - cout << ? << ? << … << blue << endl;

Simple Results

```
Slighten@SlightenCheng ttys000-bash 03:12 ~/codes/c++/hw2 $ ./a.out
Red = 0
Green = 0
Blue = 0
RGB is #000000
Slighten@SlightenCheng ttys000-bash 03:14 ~/codes/c++/hw2 $ ./a.out
Red = 1
Green = 2
Blue = 3
RGB is #010203
Slighten@SlightenCheng ttys000-bash 03:14 ~/codes/c++/hw2 $ ./a.out
Red = 255
Green = 255
Blue = 255
RGB is #FFFFFF
Slighten@SlightenCheng ttys000-bash 03:14 ~/codes/c++/hw2 $ ./a.out
Red = 255
Green = 254
Blue = 253
RGB is #FFFEFD
Slighten@SlightenCheng ttys000-bash 03:14 ~/codes/c++/hw2 $ ./a.out
Red = 200
Green = 150
Blue = 100
RGB is #C89664
Slighten@SlightenCheng ttys000-bash 03:14 ~/codes/c++/hw2 $
```