第五週

王子銜, 陳毅軒, 吳尚龍

電機通訊程式設計

March 18, 2024

Outline

介紹

Address，指的是變數在記憶體中的位址

Address

可以想像成地址的概念:

- 嘉義縣民雄鄉大學路一段 168 號就是中正大學的 address

而每個變數，在電腦上都會有自己的 address。

Address

變數有可能是相同的，但 address 必定是唯一的

Address

地名也可能是相同的，但地址卻必定是唯一的:

- 台灣有很多條中正路 (總共 316 條，from wiki)，但是每一個縣市只有一條中正路

Address

變數名稱	內容	address 記憶體位置
a	10	0x62eb3ffb3c
	↓	
中正大學	鳳梨	嘉義縣民雄鄉 大學路一段 168 號

16 進位極簡略介紹

Table: 以十進位爲例，由 10 個基本的數字所表示出來

0	1	2	3	4	5	6	7	8	9

Table: 十六進位，也理所當然的由 16 個基本的數字組成

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

16 進位極簡略介紹

A	10
B	11
C	12
D	13
E	14
F	16

- 正如同 437(10) 可以拆分爲
  $4x10^2 + 3x10^1 + 7x10^0$
- FBC(16) 可以依樣畫葫蘆地拆分成
  $F(15)x16^2 + B(11)x16^1 + C(12)x16^0$
- 其餘加減乘除的運算方式，皆和 10 進位的方法一致。

取址運算子 &

& 運算子會回傳該變數的 address，簡單來說，我們只需要 & 變數，就能得到該變數的 address
另外，如果要正確輸出位址，使用"%p"。

```c
#include<stdio.h>
int main(){
int pointer;
  printf("pointer address is %p\n", &pointer);
}
```

```
//output
pointer address is 0x7fff351ae274
```

宣告指標的方式

指標的宣告方式爲

- 所存 address 上的變數型態 *(指標名稱)

以下面這個例子來舉例:

```
1  #include<stdio.h>
2  int main(){
3    int i;
4    int *i_ptr = &i;
5    double d;
6    double *d_ptr = &d;
7    char c;
8    char *c_ptr = &c;
9  }
```

宣告指標的風格

宣告指標

1. int* p;
2. int *p;

這兩種宣告方式都是合法的，但是可讀性上第二種比較好。
將 * 放在宣告的變數型態旁，會誤導讀者以爲 int* 是整個述句的型別，但其實並不是。
以下舉例:

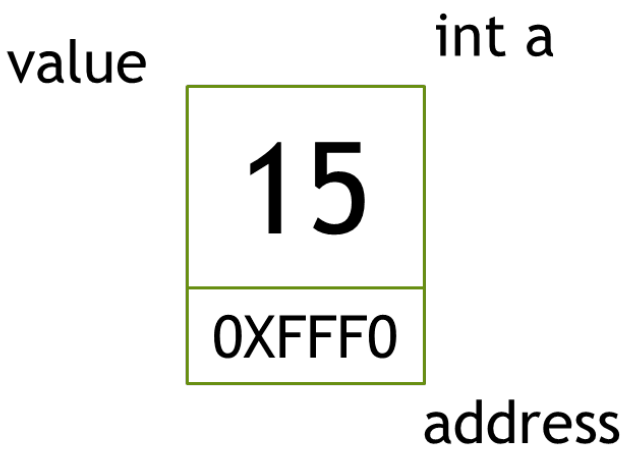
```
1  #include<stdio.h>
2  int main(){
3    int* p1,p2;/*宣告一個指標p1，以及int p2 */
4    int *p1,*p2;/*宣告兩個指標p1與p2 */
5  }
```

Pointer

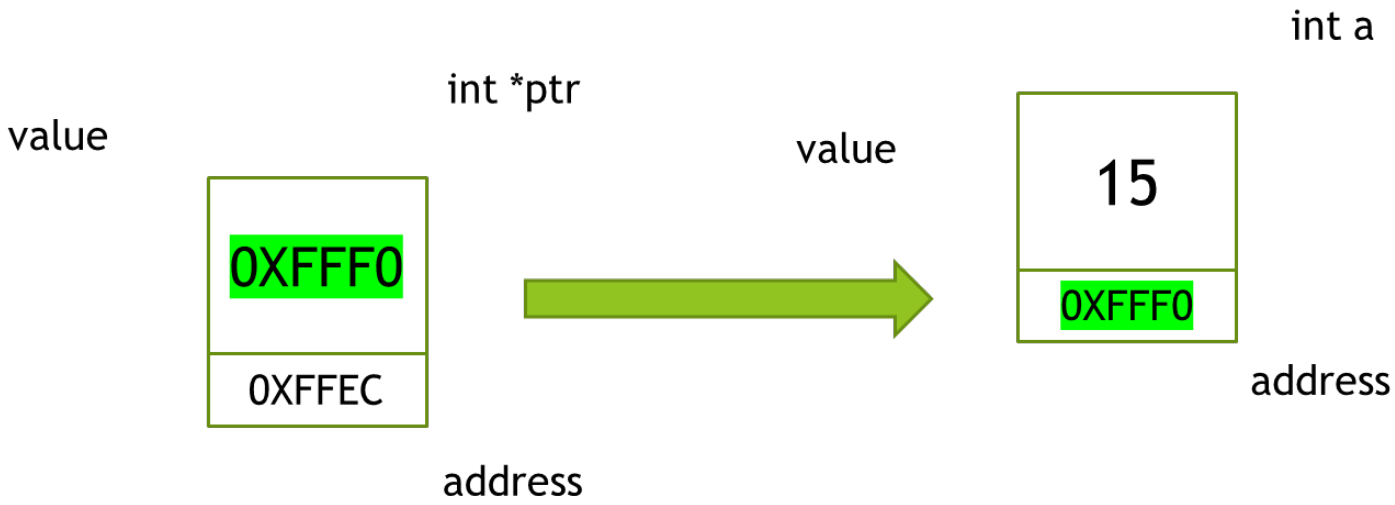
就如同我們需要設置變數，把一些數字儲存下來，我們也需要有變數能來儲存 address，而能拿來儲存 address 的變數就是指標，也就是 pointer。

```
1  #include<stdio.h>
2  int main(){
3    int a = 15;
4    int *ptr = &a;
5  }
```

設立一個名爲 ptr 的指標變數去儲存 FFF0 這個 address。



pointer 與 address 的關係



簡而言之，指標就是一個變數，用來儲存其他變數的 address

* 運算子

在這裡 * 運算子不是乘法，是取值的運算子。
從上一頁的關係圖中可以藉由 * 運算子，取得 a 變數所儲存的數值。

```
1  printf("pointer address is %p\n", ptr);
2  printf("value is %d",*ptr);
```

```
1  //output
2  pointer address is 0XFFF0
3  value is 15
```

* 運算子的意思，就是取出指標所存的那個 address 裡，所儲存的變數，
我們取出 ptr 所儲存的 adress 的數字，就可以取出變數 a 的 value 了

複習: 不同型態的位元數

變數型態	bit	byte
short	16	2
int	32	4
long	32	4
long long	64	8
float	32	4
double	64	8
char	8	1

指標與陣列的關係

陣列事實上也是指標的一種應用，不同的是，陣列是固定長度的記憶體區塊。
而指標是一個變數，用來記錄所指變數的 address。

指標與陣列的關係

```
1  int a[5] = {4, 8, 7, 6, 3};
```


```
1  printf("list address is %p\n",a);
2  printf("list first number address is %p\n",&a[0]);
```

```
1  //output
2  list address is 0XFFE0
3  list first number address is 0XFFE0
```

指標與陣列的關係

觀察到範例裡，陣列中每個元素的指標，每個元素的 address 他們所相差的都剛好是 4。
因爲陣列其實是一串相連起來的記憶體，我們可以這樣理解，我們每宣告一次陣列，編譯器會先計算出這個陣列所需要的記憶體大小，然後將這段記憶體分配給這個陣列。
一個 int a[5] 總共使用了 20bytes 的記憶體空間。因爲一個 int 變數會占用 4 bytes，而 a 是一個可儲存五個整數的陣列。
也可以理解成，自 a 指標開始，在數線上往右 20 的記憶體位置，都是被此陣列占用的。

address 的加減

address 的加減會根據資料型別的不同，去做加減法。
以範例 int a[5] 爲例:

```
1  printf("list address is %p\n",a);
2  printf("list second number address is %p\n",&a[1]);
3  printf("list second number address is %p\n",a+1);
```

```
1  //output
2  list address is 0XFFE0
3  list second number address is 0XFFE4
4  list second number address is 0XFFE4
```

陣列裡 [] 的運作模式

當我們在執行，a[3]，這個操作時，其實就等同於 *(a + 3)
a 代表的是 a[0] 的 address，也就是整個陣列的起始點而如果用中括號這個運算子，
對電腦而言，就會去操作 (a + 中括號裡的數字) 的那個 address。以範例 int a[5] 爲例:

```
1  int a[5] = {4, 8, 7, 6, 3};
2  printf("list address is %p\n",a);
3  printf("list second number address is %p\n",a+1);
4  printf("list second number is %d\n",*(a+1));
```

```
1  //output
2  list address is 0XFFE0
3  list second number address is 0XFFE4
4  list second number is 8
```

雙重指標

指標可以拿來儲存其他變數的 address，很顯然的，指標也是一個變數。
那既然指標也是一個變數，那在電腦裡，這個變數也會有自己的 address。
雙重指標的意思，就是一個儲存「指標的 ***address***」的指標。
不同於先前說過的指標，他有不同的宣告方法。

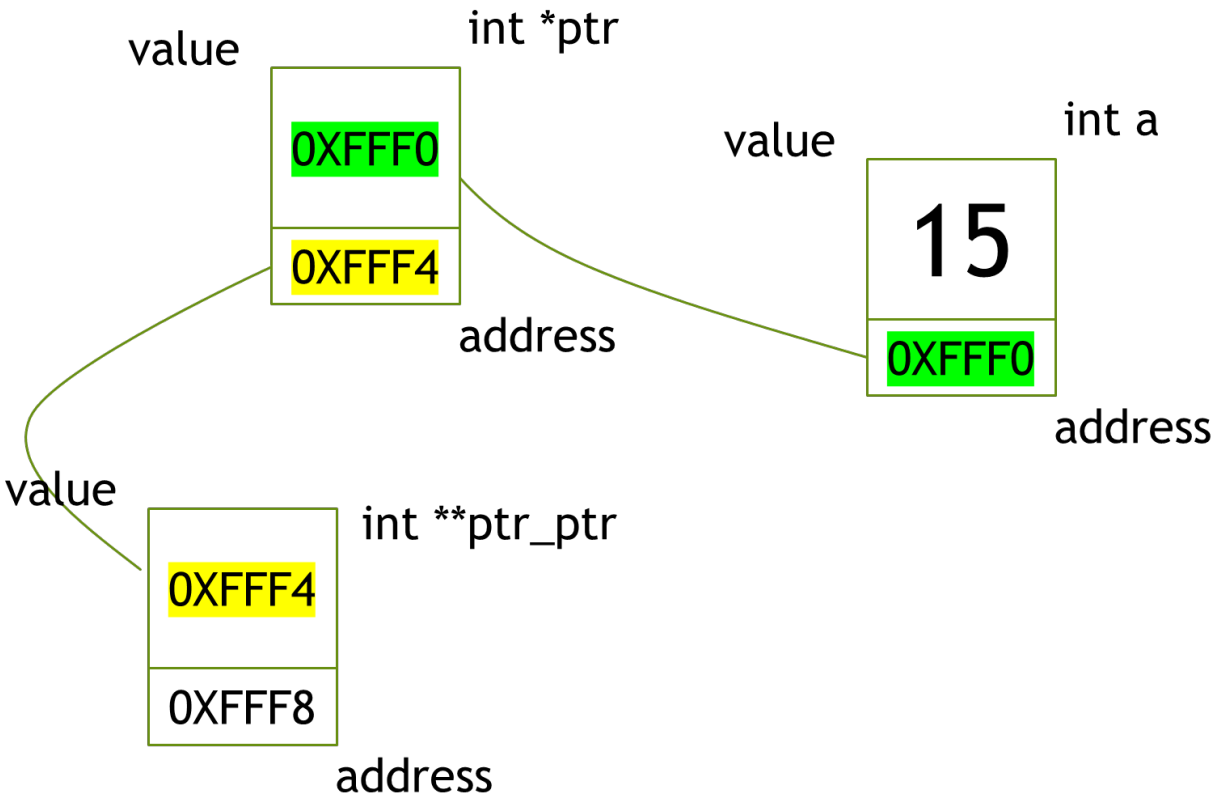
雙重指標

宣告雙重指標

- 所存 pointer 裡的 address 上的變數型態 **(雙重指標名稱)

```
1  int **ptr_ptr;//宣告雙重指標
```

雙重指標


value
int *ptr
0XFFF0
0XFFF4
address
value
int a
15
0XFFF0
address
value
int **ptr_ptr
0XFFF4
0XFFF8
address

雙重指標

```
1  int a = 15;
2  int *ptr = &a;
3  int **ptr_ptr = &ptr;
4  printf("ptr address is %p\n",ptr_ptr);
5  printf("ptr value is %p\n",*ptr_ptr);
6  printf("a value is %p",**ptr_ptr);
```

```
1  //output
2  ptr address is 0XFFF4
3  ptr value is 0XFFF0
4  a value is 15
```

動態記憶體分配

因爲 c 語言是「編譯型語言」(Compiled Languages)，也就是會先用編譯器編譯完後才作執行。
所以當我們需要動態要求一塊記憶體長度時，需要向記憶體特別要求記憶體空間。
一般陣列宣告需要給定陣列大小，譬如 int list[10]，但是在 C99 之後可以使用變數作爲宣告的參數:list[n]，使撰寫更爲方便。
但是在某些特定的狀況還是需要動態的取得記憶體。

malloc

malloc 使用方式

- void *malloc(size_t size);

- malloc 定義在 stdlib.h 函式庫裡面。
- 使用 malloc 會得到一個指標 (pointer)，指向記憶體空間。
- 因爲 malloc 回傳的是 void pointer，使用上會在前面指定型別，譬如 (int*)、(char*)、(double*)

sizeof

sizeof 使用方式

- sizeof(變數);

- 使用 sizeof 會得到一個變數大小 (byte) 的數值。

malloc

- 宣告變數:

```
1  int *iptr = (int*)malloc(sizeof(int));
2  char *cptr = malloc(sizeof(char));
3  int **ptr_iptr = (int**)malloc(sizeof(int*));
```

- 宣告一維陣列:

```
1  int *iptr = (int*)malloc(sizeof(int)*n);
2  char *cptr = malloc(sizeof(char)*n);
```

- 宣告二維陣列: 這裡留給大家自己想一下，如何動態宣告二維陣列。

本週作業加分及規定

本週作業鼓勵大家練習指標，所以本週加分方式如下:

規則

- 若程式碼中出現中括號，該題 0 分
- 若最終作業成績爲 100，總成績 +0.5 分
- 若最終作業成績爲 100，且在今日完成並給助教檢查，+1 分

檢查使用指標的方式: 我們會檢查程式碼中是否有使用中括號 [、]，不論是註解還是程式碼內都不能有這兩格符號，建議同學使用 ctrl+F 檢查是否有這兩種符號。

本週作業附錄

本週作業需要用到字串，以下提供基本使用語法:

字串

假設有一字元陣列 str，

- 使用字串，需要使用 <string.h> 函數庫
- 讀取:scanf("%s",str);
- 輸出:printf("%s",str);
- 字串長度:strlen(str);

本週作業附錄

本週作業需要用到 ASCII code，以下提供基本使用語法:

ASCII code

假設有一字元 c，

- 使用整數輸出，可以得到對應的 ASCII code 碼。printf("%d",c);
- 可以字元對減，會得到轉換爲 ASCII code 後，相減的數值。
- 使用 printf("%c", 整數);，會輸出對應 ASCII code 的字元。

本週作業附錄

ASCII code 對應表:

Control Characters				Graphic Symbols											
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	'	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	−	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F