基礎程式設計技巧(二)
程式與結構

許胖

板燒高中

February 10, 2015

大綱

大綱

溢位

問題

執行這段程式碼之後，會發生什麼現象？

溢位

問題

執行這段程式碼之後，會發生什麼現象？

```
int x = 2147483647;
cout << x + 1 << endl;
```

溢位

問題

執行這段程式碼之後，會發生什麼現象？

```
int x = 2147483647;
cout << x + 1 << endl;
```

答案

- -2147483648

溢位

問題

執行這段程式碼之後，會發生什麼現象？

```
int x = 2147483647;
cout << x + 1 << endl;
```

答案

- -2147483648
- 爲什麼？

溢位

問題

執行這段程式碼之後，會發生什麼現象？

```
int x = 2147483647;
cout << x + 1 << endl;
```

答案

- -2147483648
- 爲什麼？Ans:溢位現象

溢位

說明

溢位

說明

	01111111	11111111	11111111	11111111
+	00000000	00000000	00000000	00000001
	10000000	00000000	00000000	00000000

溢位

說明

	01111111	11111111	11111111	11111111	2147483647
+	00000000	00000000	00000000	00000001	1
	10000000	00000000	00000000	00000000	-2147483648

溢位

說明

	01111111	11111111	11111111	11111111	2147483647
+	00000000	00000000	00000000	00000001	1
	10000000	00000000	00000000	00000000	-2147483648

觀念

- 只要存資料的記憶體是有限的，那麼

溢位

說明

	01111111	11111111	11111111	11111111	2147483647
+	00000000	00000000	00000000	00000001	1
	10000000	00000000	00000000	00000000	-2147483648

觀念

- 只要存資料的記憶體是有限的，那麼
  - 儲存的資料就有範圍上的限制

溢位

說明

	01111111	11111111	11111111	11111111	2147483647
+	00000000	00000000	00000000	00000001	1
	10000000	00000000	00000000	00000000	-2147483648

觀念

- 只要存資料的記憶體是有限的，那麼
  - 儲存的資料就有範圍上的限制
  - 不然，就是精確度的限制

資料型態

資料型態	位元組	下界	上界
`char`	1	-128	127
`short`	2	-32768	32767
`int`	4	-2147483648	2147483647
`long long`	8	-9223372036854775808	9223372036854775807
`float`	4	-3.40282×10^{38}	3.40282×10^{38}
`double`	8	$-1.79769 * 10^{308}$	$1.79769 * 10^{308}$

Table: 資料型態上下界

大綱

重點回顧

重點回顧

- 位元 (bit, b)：計算機儲存資料的基本單位，只儲存 0 和 1

重點回顧

- 位元 (bit, b)：計算機儲存資料的基本單位，只儲存 0 和 1
- 位元組 (byte, B)：因爲位元很多，所以我們把 8 個位元「打包起來」，變成一個位元組

01001010

Table: 位元組

重點回顧

- 位元 (bit, b)：計算機儲存資料的基本單位，只儲存 0 和 1
- 位元組 (byte, B)：因爲位元很多，所以我們把 8 個位元「打包起來」，變成一個位元組

01001010

Table: 位元組

- 常見應用

重點回顧

- 位元 (bit, b)：計算機儲存資料的基本單位，只儲存 0 和 1
- 位元組 (byte, B)：因爲位元很多，所以我們把 8 個位元「打包起來」，變成一個位元組

01001010

Table: 位元組

- 常見應用
  - KB、MB、GB、TB、PB：資料大小

重點回顧

- 位元 (bit, b)：計算機儲存資料的基本單位，只儲存 0 和 1
- 位元組 (byte, B)：因爲位元很多，所以我們把 8 個位元「打包起來」，變成一個位元組

01001010

Table: 位元組

- 常見應用
  - KB、MB、GB、TB、PB：資料大小
  - Kbps、Mbps、Gbps：資料傳輸速度

記憶體分類

根據用途來分類，可分爲 ...

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體
   - 程式執行的地方

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體
   - 程式執行的地方
   - 我們說的記憶體、RAM 多是指這裡

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體
   - 程式執行的地方
   - 我們說的記憶體、RAM 多是指這裡
   - 關機後資料就消失

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體
   - 程式執行的地方
   - 我們說的記憶體、RAM 多是指這裡
   - 關機後資料就消失
3. 快取記憶體

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體
   - 程式執行的地方
   - 我們說的記憶體、RAM 多是指這裡
   - 關機後資料就消失
3. 快取記憶體
4. 暫存器

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體
   - 程式執行的地方
   - 我們說的記憶體、RAM 多是指這裡
   - 關機後資料就消失
3. 快取記憶體
4. 暫存器
   - 這兩個與 CPU 有關，可以先不管

記憶體分類

根據用途來分類，可分爲 ...

1. 硬碟，速度最慢
   - 長時間保存資料
   - 容量大
2. 主記憶體，俗稱記憶體
   - 程式執行的地方
   - 我們說的記憶體、RAM 多是指這裡
   - 關機後資料就消失
3. 快取記憶體
4. 暫存器
   - 這兩個與 CPU 有關，可以先不管

主軸

- 我們將集中記憶體和程式之間的關係來討論。

位址

記憶體 ...

位址

記憶體 ...

- 記憶體就像「土地」一樣，可以分配用途

位址

記憶體 ...

- 記憶體就像「土地」一樣，可以分配用途
  - 例如 `int`、`double` 等分配方式

位址

記憶體 ...

- 記憶體就像「土地」一樣，可以分配用途
  - 例如 `int`、`double` 等分配方式
- 既然要把土地分配，那麼每一塊地都要有個類似「地址」(地籍) 的編號

位址

記憶體 ...

- 記憶體就像「土地」一樣，可以分配用途
  - 例如 `int`、`double` 等分配方式
- 既然要把土地分配，那麼每一塊地都要有個類似「地址」(地籍) 的編號
- 每個位元組都有一個地址，稱爲「位址」。

位址

記憶體 ...

- 記憶體就像「土地」一樣，可以分配用途
  - 例如 `int`、`double` 等分配方式
- 既然要把土地分配，那麼每一塊地都要有個類似「地址」(地籍) 的編號
- 每個位元組都有一個地址，稱爲「位址」。

觀念

- 位元組是計算機中，擁有位址的最小單位。

觀察位址

取址運算子

下列程式會印出變數 x 的位址。

觀察位址

取址運算子

下列程式會印出變數 x 的位址。

```
#include <iostream>
using namespace std;
int main() {
  int x;
  cout << &x << endl;
}
```

觀察位址

取址運算子

下列程式會印出變數 x 的位址。

```
#include <iostream>
using namespace std;
int main() {
  int x;
  cout << &x << endl;
}
```

註

- 印出來是一個 16 進位的數字，代表變數被分配到的「位址」。

觀察位址

取址運算子

下列程式會印出變數 x 的位址。

```
#include <iostream>
using namespace std;
int main() {
  int x;
  cout << &x << endl;
}
```

註

- 印出來是一個 16 進位的數字，代表變數被分配到的「位址」。
- 分配到的位址會依據機器、作業系統、編譯器不同而不同。

觀察位址

取址運算子

下列程式會印出變數 x 的位址。

```
#include <iostream>
using namespace std;
int main() {
  int x;
  cout << &x << endl;
}
```

註

- 印出來是一個 16 進位的數字，代表變數被分配到的「位址」。
- 分配到的位址會依據機器、作業系統、編譯器不同而不同。
- 總之就是不一定啦！(ノ ° □°) ノ ⌒ ┻┻

連續宣告

觀察

觀察一下多變數的位址。

連續宣告

觀察

觀察一下多變數的位址。

```
int x, y, z;
cout << &x << endl << &y << endl << &z << endl;
```

連續宣告

觀察

觀察一下多變數的位址。

```
int x, y, z;
cout << &x << endl << &y << endl << &z << endl;
```

結果

- 以筆者的機器來說，會出現以下結果：

連續宣告

觀察

觀察一下多變數的位址。

```
int x, y, z;
cout << &x << endl << &y << endl << &z << endl;
```

結果

- 以筆者的機器來說，會出現以下結果：

```
0040EC64
0040EC68
0040EC6C
```

連續宣告

觀察

觀察一下多變數的位址。

```
int x, y, z;
cout << &x << endl << &y << endl << &z << endl;
```

結果

- 以筆者的機器來說，會出現以下結果：

0040EC64 ← 變數 x 的位址
0040EC68 ← 變數 y 的位址
0040EC6C ← 變數 z 的位址

連續宣告

觀察

觀察一下多變數的位址。

```
int x, y, z;
cout << &x << endl << &y << endl << &z << endl;
```

結果

- 以筆者的機器來說，會出現以下結果：

  0040EC64 ← 變數 x 的位址
  0040EC68 ← 變數 y 的位址
  0040EC6C ← 變數 z 的位址

- 以筆者的機器，int 的大小是 4 個位元組。

大印第安和小印第安

問題

假設現在 `int x = 0x12345678;` 的位址是 0040EC64，那麼這個 16 進位的數字「實際上」是怎樣儲存的呢？

記憶體分配

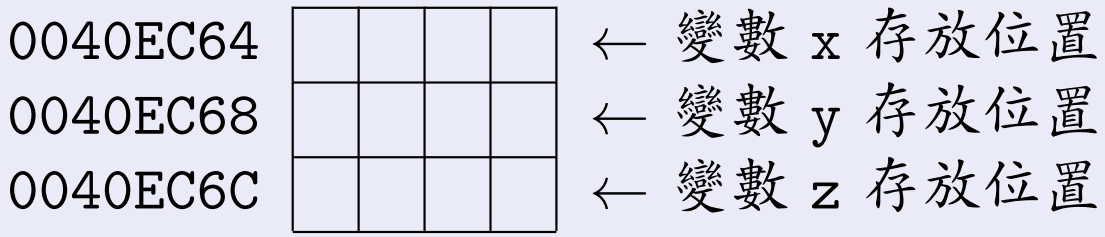
說明

- 觀察這幾個記憶體

記憶體分配

說明

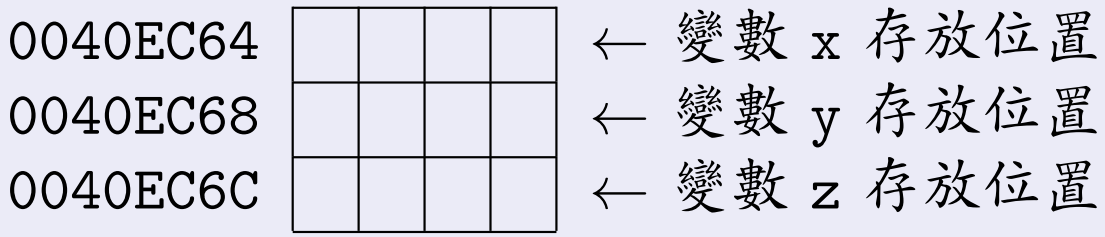
- 觀察這幾個記憶體



記憶體分配

說明

- 觀察這幾個記憶體



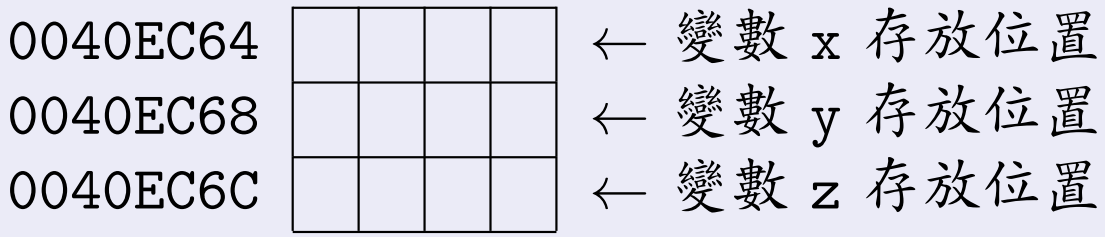
註

- x 佔 4 個位元組 (0040EC64、0040EC65、0040EC66、0040EC67)

記憶體分配

說明

- 觀察這幾個記憶體



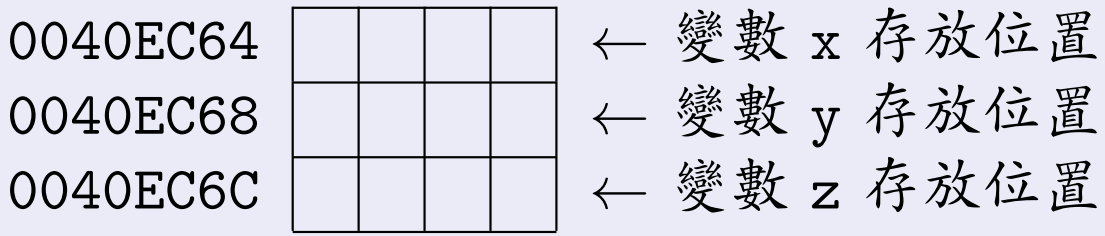
註

- x 佔 4 個位元組 (0040EC64、0040EC65、0040EC66、0040EC67)
- y 佔 4 個位元組 (0040EC68 到 0040EC6B)

記憶體分配

說明

- 觀察這幾個記憶體



註

- x 佔 4 個位元組 (0040EC64、0040EC65、0040EC66、0040EC67)
- y 佔 4 個位元組 (0040EC68 到 0040EC6B)
- z 佔 4 個位元組 (0040EC6C 到 0040EC6F)

指標

概念

- 像是一根「手指」，指著一塊記憶體。

指標

概念

- 像是一根「手指」，指著一塊記憶體。
- 指著的記憶體，可以指定用途。

指標

概念

- 像是一根「手指」，指著一塊記憶體。
- 指著的記憶體，可以指定用途。

宣告

```
int *x;
```

指標

概念

- 像是一根「手指」，指著一塊記憶體。
- 指著的記憶體，可以指定用途。

宣告

```
int *x;
```

註

- x 是一個指標，會指向一塊記憶體。

指標

概念

- 像是一根「手指」，指著一塊記憶體。
- 指著的記憶體，可以指定用途。

宣告

```
int *x;
```

註

- x 是一個指標，會指向一塊記憶體。謎之音：「怎麼指？」

指標

概念

- 像是一根「手指」，指著一塊記憶體。
- 指著的記憶體，可以指定用途。

宣告

```
int *x;
```

註

- x 是一個指標，會指向一塊記憶體。謎之音：「怎麼指？」
  - 去儲存某一塊記憶體的位址。

指標

概念

- 像是一根「手指」，指著一塊記憶體。
- 指著的記憶體，可以指定用途。

宣告

```
int *x;
```

註

- x 是一個指標，會指向一塊記憶體。謎之音：「怎麼指？」
  - 去儲存某一塊記憶體的位址。
- 那塊記憶體的用途是 int。

指標實戰

實戰

```
int x = 5;
int *ptr = &x;
cout << x << endl;
cout << *ptr << endl;
```

大綱

練習題

UVa 10035 - Primary Arithmetic

不難。

練習題

UVa 10035 - Primary Arithmetic

不難。

UVa 10038 - Jolly Jumpers

讀懂題意就不難。

練習題

UVa 10035 - Primary Arithmetic

不難。

UVa 10038 - Jolly Jumpers

讀懂題意就不難。

UVa 109 - SCUD Busters

這一題敘述和公式稍嫌複雜，但是用我們之前所學的工具依然可以輕鬆解決。