Data Structure and Algorithm

Quiz #1, Solution

Tuesday, February 26, 2013

評分標準只針對本試題，以 100 分計，點名分另外計算。而本試卷的分數對學期成績影響不大，煩請同學對分數不要太過計較。

本試卷的宗旨在於點出同學對 pointer 不熟的部分，希望能讓同學在實際 coding 的時候可以少走些冤枉路 (崩潰的 debug 之路)。

不管有無理由，如果答案正確便給全部分數。
答案錯誤的話，如果有寫理由，則會依觀念給分。如果沒寫理由，但答案切題，仍因為勇於嘗試會給予 4 分。

Problem 1. Please write down the output for each `printf()`.

Note that the code may fail to be compiled or the execution may not happen in an expected way. In these cases, you should explain why you think it will not work.

1.

```
1 int *a, *b;
2 int* c, d;
3 a = 1; b = 1; c = 1; d = 1;
4 printf("%d\n", a+b);
5 printf("%d\n", c+d);
```

兩個 printf 各 10 分。

首先是宣告部分，a, b, c 都是 pointer of integer，d 是 integer。星號放置的位置不會影響結果。

第 3 行的部份，在型別嚴格判定的 C++ 中，會有型別錯誤 (`int*` 不能轉換成 `int`)，在 C 語言中則是可以這樣使用。

`printf()` 用`%d` 輸出位址，雖然使用上不太正確，但是在 pointer 大小為 4 bytes 之電腦上仍可正常輸出記憶體位址。當初考量到同學可能對`%p` 的行為不熟，才如此設計題目。
(如果指出應使用`%p` 輸出指標，給予 bonus 2 分)

$a + b$ 是 `int*` + `int*`。(型別判斷 2 分)
這個加法沒有意義 (2 分)，且會編譯錯誤 (2 分)。

$c + d$ 是 `int*` + `int`。(型別判斷 2 分)
這個加法是指根據原位址所做的位移，這邊是位址 1 往後位移 1 個 `int` 長度後的位址 (2 分)，是 $1 + 1 \times$ `sizeof(int)`。(2 分)
由於題目沒有定義 `sizeof(int)`，故合理的 `sizeof(int)` 之值都視為正確。常見的答案是 5 (4 bytes 的 int)。

2.

```
1 char ary[] = "DSA is so fun.";      //strlen(ary) is 14
2 char *ptrChar = ary;
3 int *ptrInt = (int*) ary;
4 printf("%d\n", sizeof(ary));
5 printf("%s %s\n", ptrChar+1, ptrInt+1);
6 *ptrChar++;
7 printf("%c %s\n", *ptrChar, ary);
```

三個 printf 各 10 分。

第 4 行，`sizeof(ary)` 是整個陣列的大小。(2 分)
宣告 `ary` 的時候會多分配一格來儲存`'\0'`。(2 分)
所以用了 `(strlen(ary) + 1) × sizeof(char)` 的空間。常見的答案是 15。(2 分)

第 5 行，`char*` 位移 1 個 bytes (1 分)，所以答案是 “`SA is so fun.`”。(1 分)
`int*` 位移 4 個 bytes (2 分)，所以答案是 “`is so fun.`”。(2 分)

第 6, 7 行，“`*`” 運算 (對指標取值) 的優先度比較低，於是 “++” 會先計算。(2 分)
所以 “++” 會對指標本身做運算，然後把位址給 “`*`” 運算，並在這行結束才把指標位址加 1。(2 分)
最後指標會指到第二個字，字串本身沒有被修改。分別輸出 “`S`”、“`DSA is so fun.`”(2 分)

3.

```
1 int *square(int n){
2     int result = n * n;
3     return &result;
4 }
5 int main(){
6     int *ptr = square(10);
7     printf("%d\n", *ptr);
8 }
```

本題 10 分。

第 6 行是宣告一個指標，並指向從 `square()` 傳回的記憶體位址。第 7 行是輸出 `ptr` 指向位址所儲存的值。而 `ptr` 指向的位址是 `result` 的位址，所以 `*ptr` 便是 `result` 的值。

首先有些同學認為程式碼有多餘或是遺漏 “`*`”，但這份程式碼沒有任何語法錯誤。(2 分)
能預期輸出結果是 “100”。(2 分)
但是 `result` 變數是 “區域” 變數，在 `square` 函式結束時便被釋放了，該位置可能會被拿來做其他用途，於是取用該記憶體位置的結果是無法預期且可能造成記憶體錯誤。(2 分)

Problem 2. The following is part of a C program.

```
1 int n = 10;
2 int A[n];
3 int *B = (int*) malloc( sizeof(int) * n );
```

1. The sizes of array A and array B are the same. Please describe the difference between them. The program may crash when n becomes very large. Why?
   (Hint: focus on array A.)

   本題 20 分。

   第 2 行，A 陣列的宣告，在編譯器術語叫做 variable length array (VLA)。也就是可以用變數控制宣告的靜態陣列大小。

   原先陣列的被規定必須能在編譯時期就被計算出來，換句話說便是必須是個常數。但 VLA 在 1999 年新制定的標準 (c99 標準) 被放進來，甚至支援 `void f(int size, int ary[size]){}` 這樣的宣告方式。

   但 **C++ 標準中一直沒有放入 VLA**。雖然 gnu 有額外支援，所以使用 gcc/g++ 仍會編譯通過，但使用 VLA 並不符合標準的 C++ 語法，在跨環境或是使用不同編譯器可能會編譯錯誤。

   另外，即使使用 VLA，被分配仍是 stack 區的記憶體。在大多的作業系統中，系統 stack 大小會有個上限，通常不超過 10MB。而這塊記憶體是程式執行的時候便被靜態分配好的，於是使用過量的話便如同記憶體使用越界一樣，發生錯誤。

   第 3 行是一個正規的從 heap 區動態宣告記憶體，且自行管理的方式。

   除了上述兩者記憶體區域不同外。

   兩者間的差別仍有：

   - 型態的差異。(array 和 pointer)
   - 程式只要離開 A 存活的域 (scope)，便會被系統自動回收。B 需要手動管理記憶體。

   認為語法無誤。(4 分)
   提出任何 “正確” 的差異，如果錯誤不額外扣分。(6 分)
   能夠區分 A, B 記憶體放置區段的不同 (3 分)，並提出兩者的容量上限有差異 (3 分)。
   其他每個小錯誤扣 2 分。

2. How to use `malloc()` to declare a $n \times m$ array C? Please write down the C code.
   (Hint: you can use a **for loop** and `int**`.)

   本題 20 分。

   程式碼僅供參考。

```
1 int **C = (int**) malloc( n * sizeof(int*) );
2
3 for(int i=0; i<n; i++)
4     C[i] = (int*) malloc( m * sizeof(int) );
```

   對一行給 5 分。`int**` 指標的宣告佔 1 分。同樣每個大錯誤扣 5 分，小錯誤扣 2 分。

   n, m 相反不扣分，但仍請同學注意 row 和 column 的差異。