Advanced C Programming And It's Application

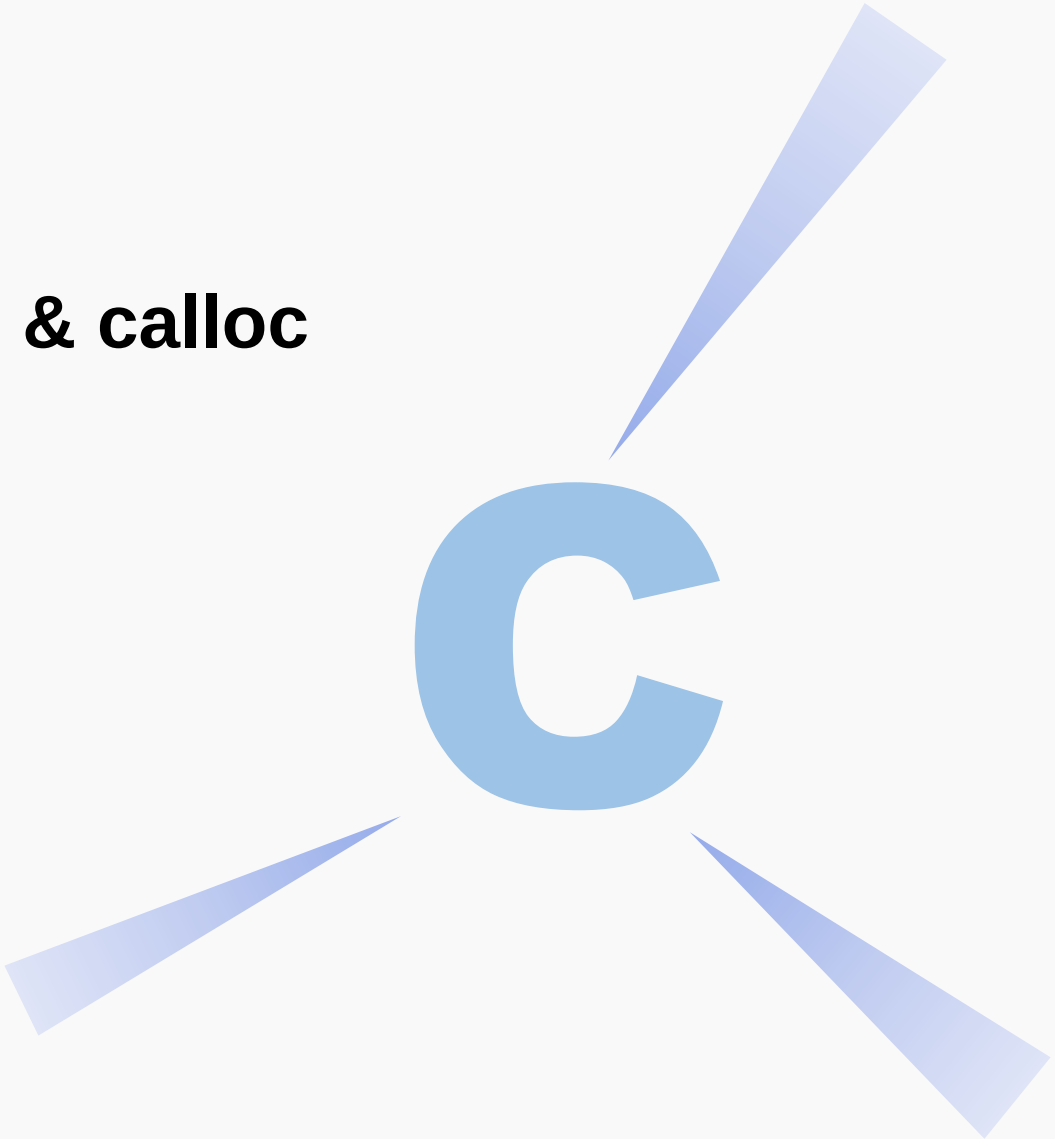
Dynamic Memory Allocation – Part II.

Assistant Prof. Chan, Chun-Hsiang

Department of Artificial Intelligence, Tamkang University

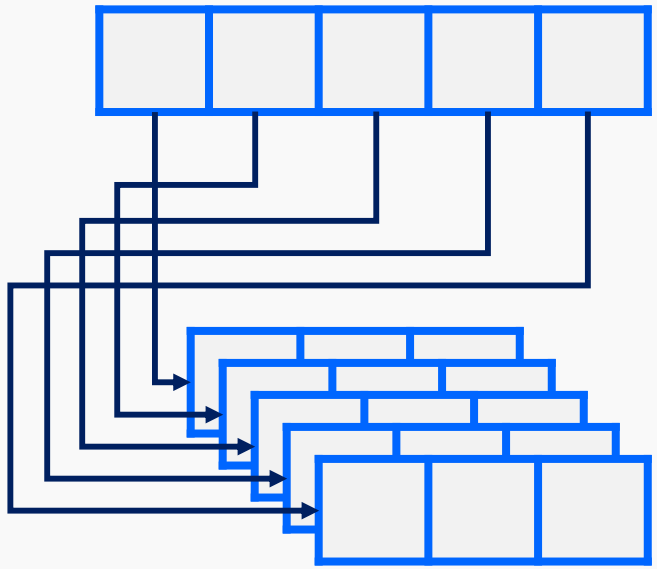
Dec. 22, 2021

大綱



Dynamic 2D Array – malloc

既然可以做到一維矩陣的動態記憶體配置，二維矩陣當然也可以做。作法就是先配置一維矩陣，再跟對一維矩陣每個element再進行配置一次一維矩陣，就可以有二維矩陣動態記憶體配置效果。提到這邊，應該可以發現一件重要的事情，因為我們要進行兩次的配置，然而malloc回傳的是一個pointer，所以如果要進行二維矩陣記憶體配置的時候，就要使用到pointer to pointer。



Dynamic 2D Array – malloc

void *malloc(size_t size)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-9: memory allocation for 2D array with malloc()
        printf("Ex 12-9: memory allocation for 2D array with malloc()n");
        int col = 4, row = 2, i, j;
        // using pointer-to-pointer
        int **arr = (int**) malloc(sizeof(int*)*row);
        // using malloc again to add memory space for column elements
        for (i=0; i<row; i++){
                arr[i] = (int*) malloc(sizeof(int)*col);
        }
```

Dynamic 2D Array – malloc

void *malloc(size_t size)

```
// print values
for (i=0; i<row; i++){
        for (j=0; j<col; j++){
                printf("(%d,%d) %8d (%p)\t", i, j, arr[i][j], &arr[i][j]);
        }
        putchar('\n');
}
// free memory space
free(arr);}
```

```
Ex 12-9: memory allocation for 2D array with malloc()
-----------after malloc------------
(0,0)   726320 (00000000000B75B0)   (0,1)        0 (00000000000B75B4)   (0,2)   721232 (00000000000B75B8)   (0,3)        0 (00000000000B75BC)
(1,0)   726320 (00000000000B75D0)   (1,1)        0 (00000000000B75D4)   (1,2)   721232 (00000000000B75D8)   (1,3)        0 (00000000000B75DC)
--------------free()---------------
```

void *calloc(size_t nitems, size_t size)

Dynamic 2D Array – calloc

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-10: memory allocation for 2D array with calloc() */
        printf("Ex 12-10: memory allocation for 2D array with calloc()\n");
        int col = 4, row = 2, i, j;
        // using pointer-to-pointer
        int **arr = calloc(row, sizeof(int*));
        // using calloc again to add memory space for column elements
        for (i=0; i<row; i++){
                arr[i] = calloc(col, sizeof(int));
        }
```

void *calloc(size_t nitems, size_t size)

Dynamic 2D Array – calloc

```
        printf("----------after calloc-----------\n");
        // print values
        for (i=0; i<row; i++){
                for (j=0; j<col; j++){
                        printf("(%d,%d) %2d (%p)\t", i, j, arr[i][j], &arr[i][j]);
                }
                putchar('\n');
        }
        printf("-------------free()--------------\n");
        // free memory space
        free(arr);
}
```

```
Ex 12-10: memory allocation for 2D array with calloc()
----------after calloc-----------
(0,0)  0 (0000000000BF75B0) (0,1)  0 (0000000000BF75B4) (0,2)  0 (0000000000BF75B8) (0,3)  0 (0000000000BF75BC)
(1,0)  0 (0000000000BF75D0) (1,1)  0 (0000000000BF75D4) (1,2)  0 (0000000000BF75D8) (1,3)  0 (0000000000BF75DC)
-------------free()--------------
```

Memory Leak

記憶體流失(memory leak)是一個蠻嚴重的問題，舉一個在維基看到的例子 – 電梯按鈕:

當按下按鈕時：

1. 要求使用記憶體，用作記住目的樓層
2. 把目的樓層的數字儲存到記憶體中
3. 電梯是否已到達目的樓層？
4. 如是，沒有任何事需要做：程式完成

否則：

1. 等待直至電梯停止
2. 到達指定樓層
3. 釋放剛才用作記住目的樓層的記憶體

Memory Leak

在C/C++中，有許多debug tool會自動檢查不能存取的記憶體，進而達到避免記憶體流失的問題。這一般會出現兩個議題:

(1) 重要機密資訊沒有隨著程式結束而釋放掉，導致駭客有機會trace到這些資料內容。

(2) 因為記憶體不斷的流失，所以所剩的記憶體空間會越來越小，最後就沒有空間了。

以下的範例，是讓大家了解常見的記憶體流失會出現在哪裡。

Memory Leak :: 01 :: forget to free

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        /*Ex 12-11: Memory Leak 01 :: forget to free memory */
        int size = 10;
        int *p = (int*) malloc(sizeof(int)*size);
        // after malloc
        printf("%10d (%p)\n", *p, &p);
        // free memory space
        // ...
}
```

```
   1729920 (000000000061FE10)
```

Memory Leak :: 02 :: allocate too large memory

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-12: Memory Leak 02 :: too large memory allocation */
        int size = 100000;
        int *p = (int*) malloc(sizeof(int)*size);
        // after malloc
        printf("Ex 12-12: Memory Leak 02 :: too large memory allocation\n");
        printf("%10d (%p)\n", *p, &p);
        // free memory space
        free(p); // safe and okay
        p = 0;
}
```

```
Ex 12-12: Memory Leak 02 :: too large memory allocation
   1442128 (000000000061FE10)
```

Memory Leak :: 03 :: free correct variable (func)

```c
#include <stdio.h>
#include <stdlib.h>

int *expand(int size){
        int *exp = (int*) malloc(sizeof(int)*size);
        return exp;
}
int main(){
        /*Ex 12-13: Memory Leak (function) 03 :: free correct variable? */
        int *arr = expand(10);
        free(arr); // Is it safe? Yes.
        arr = 0;
}
```

Memory Leak :: 04 :: free correct variable (func)

```c
#include <stdio.h>
#include <stdlib.h>

int expand(int *exp, int size){
        exp = (int*) malloc(sizeof(int)*size);
}

int main(){
        /*Ex 12-14: Memory Leak (function) 04 :: free correct variable? */
        int *arr;
        expand(arr, 10);
        free(arr); // Is it safe? No.
}
```

Memory Leak :: 05 :: free correct variable (func)

```c
#include <stdio.h>
#include <stdlib.h>

int expand(int **exp, int size){
        *exp = (int*) malloc(sizeof(int)*size);
}

int main(){
        /*Ex 12-15: Memory Leak (function) 05 :: free correct variable? */
        int *arr;
        expand(&arr, 10);
        free(arr); // Is it safe? Yes.
}
```

Common Mistake

最後我們來談一下常見的問題:

(1) 釋放記憶體之後再次呼叫

(2) 釋放兩次記憶體

(3) 釋放錯誤的變數記憶體空間

“If you love someone, set them free.”

<common mistake/>

Common Mistake :: call after free

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    /*Ex 12-16: Common Mistake 01 :: call after free */
    int size = 10;
    int *p = (int*) malloc(sizeof(int)*size);
    // free memory space
    free(p); // safe and okay
    p = 0;
    // call again
    printf("%d\n", p[1]); // occurs error
}
```


</common mistake>

Common Mistake :: free twice

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-17: Common Mistake 02 :: free twice */
        int size = 10;
        int *p = (int*) malloc(sizeof(int)*size);
        int *q = p;
        printf("%p %p\n", p, q);
        // free memory space
        free(q); // safe and okay
        free(p); // that is redundant!
}
```

```
0000000000A41A90 0000000000A41A90
```

Common Mistake :: free wrong data type

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    /*Ex 12-18: Common Mistake 03 :: free wrong type */
    int size = 10;
    int *p = (int*) malloc(sizeof(int)*size);
    // free memory space
    free(p); // safe and okay
    free(size); // ← warning: passing argument 1 of 'free' makes pointer from integer without a cast [-Wint-conversion]
}
```

作業一: 信用卡驗證機

大家平常應該都會有網路購物的習慣，一般常見的支付方式會有: ATM轉帳、信用卡一次繳清、行動支付(Apple pay, Line pay, and ...)等。那麼你有想過支付平台怎麼知道你寫的信用卡是正確的呢? 難道是後台有所有人的信用卡資料嗎? ➔ 這聽起來很沒有隱私!

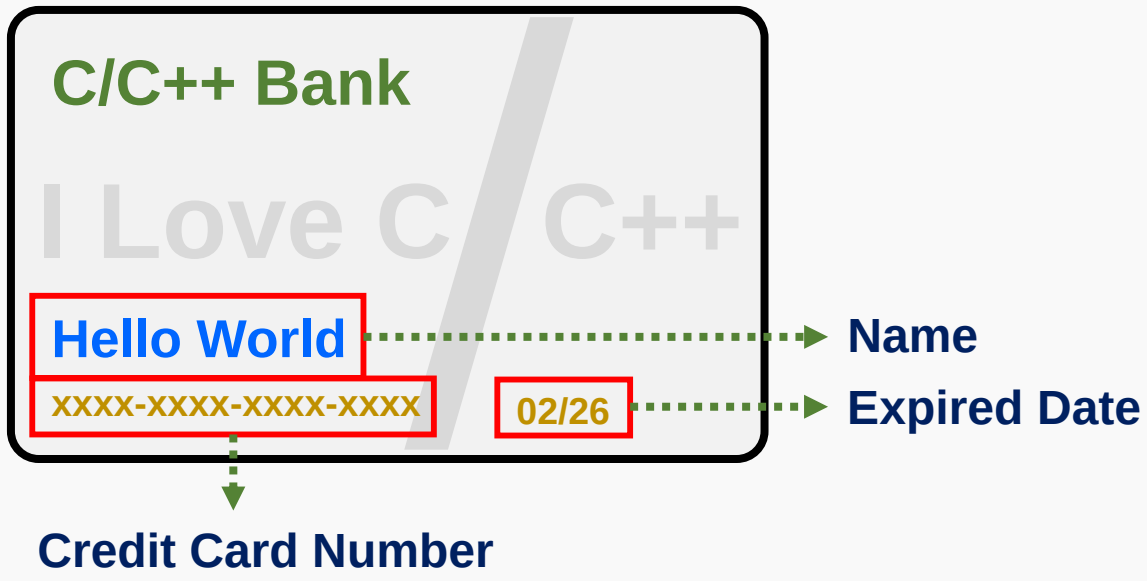
實際上，再做驗證的方法有很多，我們這邊提一個比較簡單的方式，他的計算過程與方式跟我們之前做身分證號碼驗證很類似。

To be continued…

作業一: 信用卡驗證機

既然要知道怎麼計算出來的，首先你需要一組信用卡號碼。

Credit Card Number Generator: LINK



VISA

4024-0071-9977-6666
4485-7988-5649-8808
4556-8992-6892-5091
4024-0071-6142-5789
4716-5791-5076-2821

MasterCard

5509-8198-2415-1128
5554-8159-8205-8117
5318-7690-3591-6697
5595-6642-3010-8980
5298-4825-4483-4849

JCB

3088-6767-5562-6997
3337-7088-8512-1695
3112-8128-6453-8878
3088-6456-2277-6632
3096-4349-0381-0125

AMERICAN EXPRESS

3756-1948-2665-662
3415-1949-7580-693
3406-9480-8889-710
3708-1818-3272-068
3446-6073-6603-313

作業一: 信用卡驗證機

1. 左邊數過來: 奇數的數字乘2
2. 如果大於9，則可以減9或兩位數之和
3. 左邊數過來: 偶數的數字乘1
4. 全部加總除以10，若整除則此信用卡號碼為真

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Number	5	4	4	1	9	6	3	6	9	6	1	8	6	9	4	4
Weight	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1
Check if > 9; then – 9	10	4	8	1	18	6	6	6	18	6	2	8	12	9	8	4
Result	1	4	8	1	9	6	6	6	9	6	2	8	3	9	8	4

Sum = 90 ➔ 90 ÷ 10 = 9 ➔ 可整除，故此信用卡號碼valid!

作業一: 信用卡驗證機

為了防止個資洩露，程式設計時要利用動態記憶體配置存取使用者之信用卡卡號，使用完畢之後，必須先將記憶體釋放，再將指標設為空指標(NULL Pointer)。
並且需要兩個基本防呆，還有一個發卡銀行判別功能:
(1) 長度必須為16
(2) 起始碼必須為3、4或5
(3) 依照前面所說，判定該卡由哪個發卡銀行所發行

作業一: 信用卡驗證機

```
int CreditCardNumberChecker(){
        …
        printf("Please enter your credit card number!\n");
        scanf("%…", …);
        …
        if length != 16, then print (length = %...) Incorrect Number Length!
        if starting number is not 3, 4, or 5, then print (%...) Incorrect Starting Number!
        do calculation
        recognize bank (VISA, Master Card, JCB)
        report result:
                if correct ➔ %s Credit Card Number (%s with total %d) is valid!
                if incorrect ➔ %s Credit Card Number (%s with total %d) is INvalid!
        free
}

int main(){
        printf("Welcome to use Credit Card Number Validation Program\n");
        CreditCardNumberChecker();
}
```

作業一: 信用卡驗證機

本題測試資料為:

4929196153082660
2124873248421232
3337146185414447
3337613081836067 1
4815624643465738
550515193717526
5461494070161563

*必須要截圖測資結果!

```
Welcome to use Credit Card Number Validation Program
Please enter your credit card number!
3528952455625037
JCB Credit Card Number (3528952455625037 with total 70) is valid!
```

```
Welcome to use Credit Card Number Validation Program
Please enter your credit card number!
4392178164350250
VISA Credit Card Number (4392178164350250 with total 60) is valid!
```

```
Welcome to use Credit Card Number Validation Program
Please enter your credit card number!
5221372813037468
Master Card Credit Card Number (5221372813037468 with total 61) is INvalid!
```

```
Welcome to use Credit Card Number Validation Program
Please enter your credit card number!
752213728130374 68
(strlen = 17) Incorrect Number Length!
```

```
Welcome to use Credit Card Number Validation Program
Please enter your credit card number!
7522137281303746
(7) Incorrect Starting Number!
```

參考資料

Code Part

1. https://openhome.cc/Gossip/CGossip/MallocFree.html
2. http://tw.gitbook.net/c_standard_library/index.html
3. https://ithelp.ithome.com.tw/articles/10204463
4. 蔣宗哲教授講義

Credit Card

1. https://hackmd.io/@CynthiaChuang/Check-Credit-Card
2. https://hsuchihting.github.io/javascript/20210202/711211859/