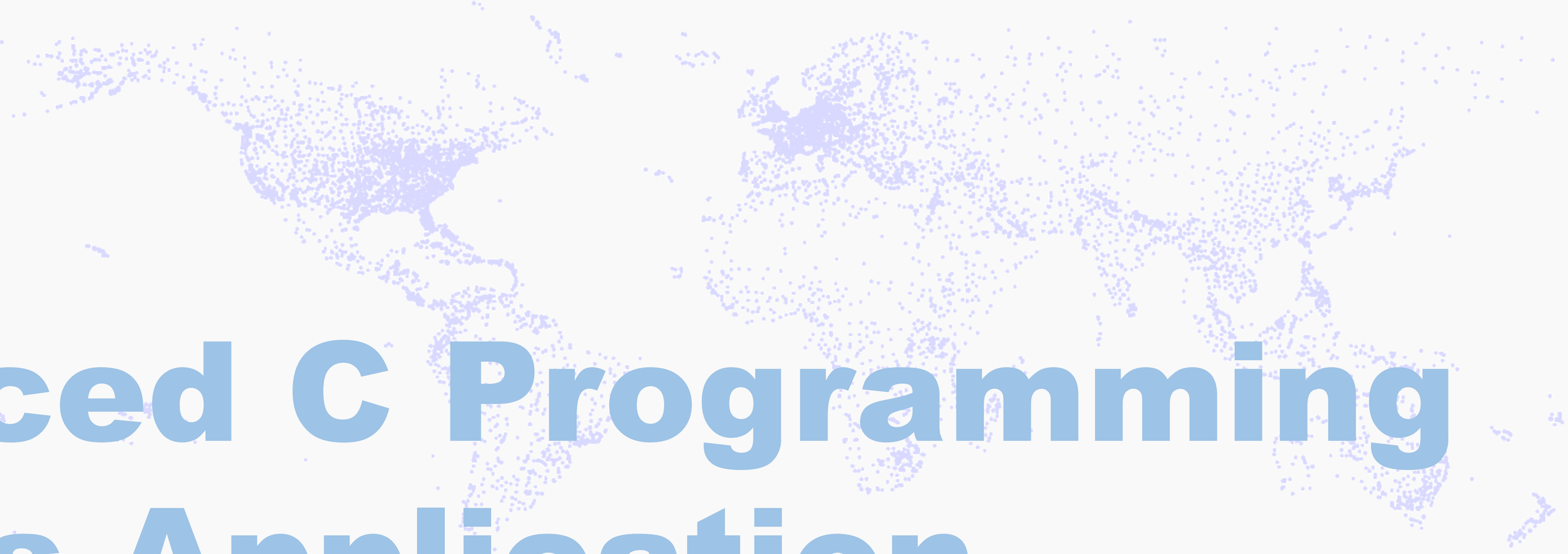


Advanced C Programming And It’s Application

Dynamic Memory Allocation – Part I.

Assistant Prof. Chan, Chun-Hsiang

Department of Artificial Intelligence, Tamkang University

Dec. 22, 2021

大綱

[1] Concept

[2] Dynamic Memory Allocation

[3] malloc

[4] free

[5] calloc

[6] Dynamic 1D Array – realloc

<Concept/>

Concept

為甚麼我們需要”動態記憶體配置”?

其實很多時候根本不知道我們究竟需要多少記憶體空間，如果今天你的老師請你寫一個code可以計算全班成績的程式碼。

你究竟會怎麼做呢?

(1) 班上有多少學生?

(2) 有幾次的成績需要輸入?

(3) 需要加權嗎? 權重為何?

你是不是覺得你問完所有問題了?

立刻開啟你的編輯器:

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

int main(){
        int numberOfStudent = 30, items = 10;
        int grade[numberOfStudent][items] = {0};
        …
}
```

</Concept>

Concept

為甚麼我們需要”動態記憶體配置”?

你有想過你的老師說不定同一門課有開好幾個班級?
每一班的人數也不一樣?
分數的計算方法也可能不一樣?

那該怎麼辦呢?

Dynamic Memory Allocation

那麼究竟要怎麼做動態記憶體配置呢?

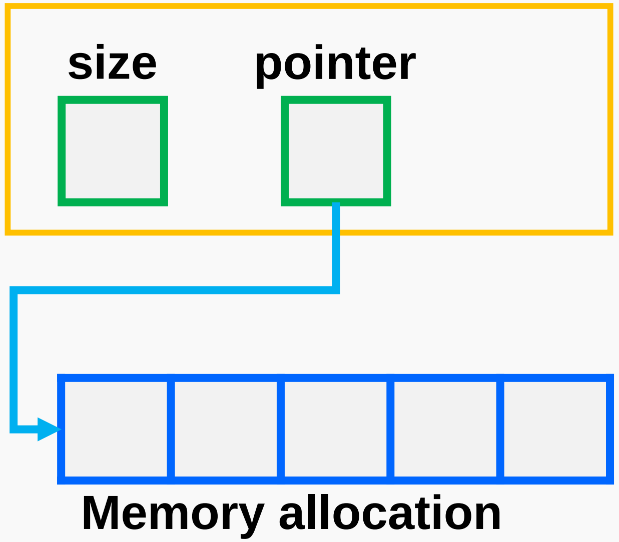
首先，利用malloc() or calloc()來動態配置所需要的記憶體空間

使用完畢記得用free()回收掉剛剛配置的記憶體空間

Function	**Meanings**
void *malloc(size_t size)	配置所需要的記憶體空間(size_t) ，並回傳一個指標
void *calloc(size_t nitems, size_t size)	配置所需要的記憶體空間(size_t)，並回傳一個指標
void *realloc(void *ptr, size_t size)	調整原先指標ptr指向已安排好的記憶體空間，並回傳一個指標
void free(void *ptr)	釋放malloc、calloc、或是realloc所配置的記憶體

malloc

我們先介紹最常用的動態記憶體配置函數malloc()，可從他的函數input argument中看到，只要輸入使用者所需的記憶體空間，以及資料型態，就可以回傳一個配置好的指標提供後續使用。



```
void *malloc(size_t size)
```

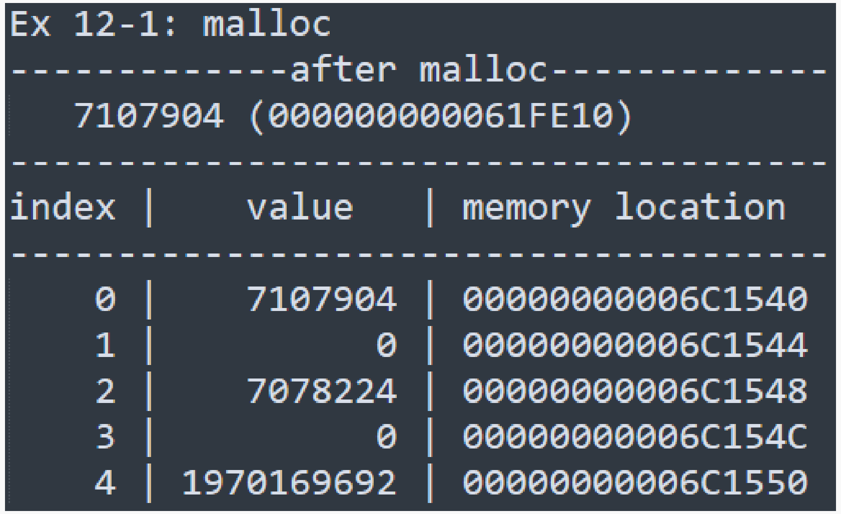
需要注意的是，這邊所配置的記憶體，不會自動將所有元素變成0，取而代之的是隨機亂數。

此外，如果在block中做動態記憶體配置的時候，配置的記憶體會隨著block結束，而結束。

malloc

void *malloc(size_t size)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-1: malloc */
        printf("Ex 12-1: malloc \n");
        int size = 5, i;
        int *p = (int*) malloc(sizeof(int)*size);
        printf("%10d (%p)\n", *p, &p); // after malloc
        // print value
        printf("index |    value   | memory location\n");
        for (i=0; i<size; i++){
                printf("%5d | %10d | %p\n", i, p[i], &p[i]);
        }
}
```

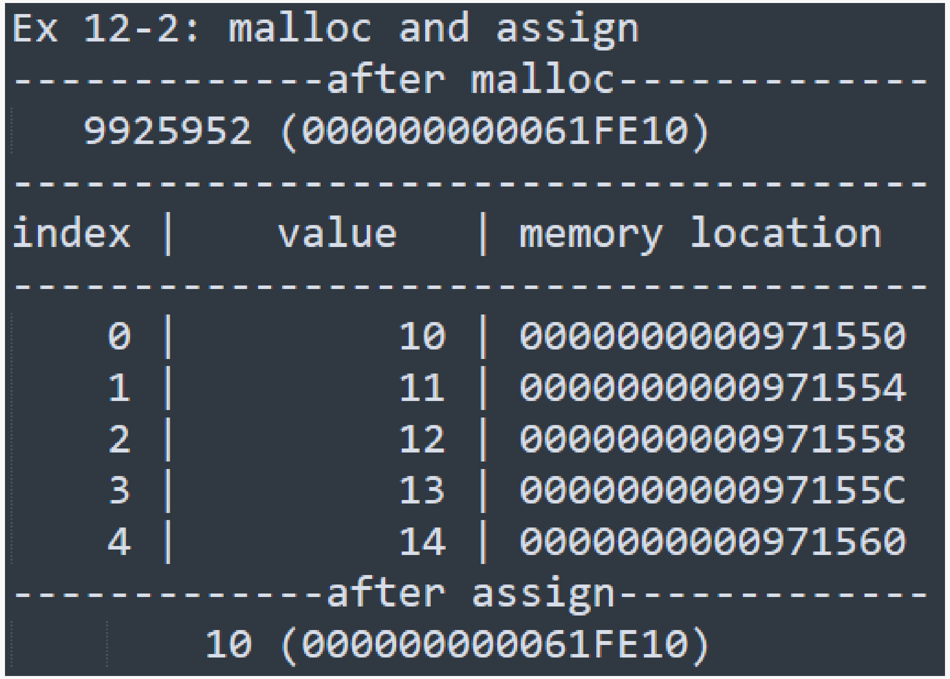


```
Ex 12-1: malloc
-------------after malloc------------
   7107904 (000000000061FE10)
-------------------------------------
index |    value   | memory location
-------------------------------------
    0 |    7107904 | 00000000006C1540
    1 |          0 | 00000000006C1544
    2 |    7078224 | 00000000006C1548
    3 |          0 | 00000000006C154C
    4 | 1970169692 | 00000000006C1550
```

malloc and assign value

void *malloc(size_t size)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-2: malloc and assign*/
        printf("Ex 12-2: malloc and assign\n");
        int size = 5, i;
        int *p = (int*) malloc(sizeof(int)*size);
        printf("%10d (%p)\n", *p, &p); // after malloc
        // assign value
        printf("index |    value    | memory location\n");
        for (i=0; i<size; i++){
                p[i] = i+10;
                 printf("%5d | %10d | %p\n", i, p[i], &p[i]);
        }
        printf("%10d (%p)\n", *p, &p);
}
```



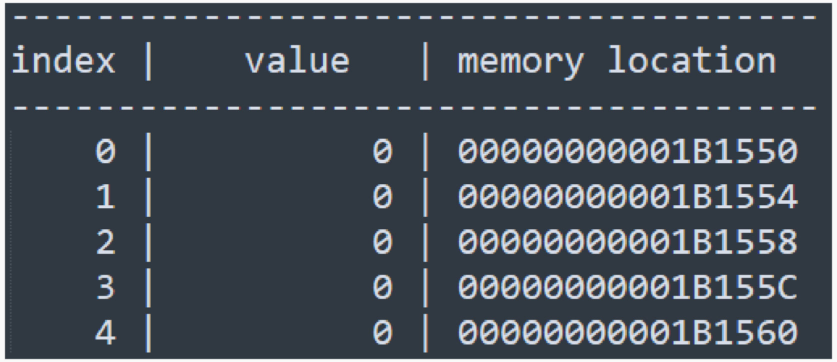
```
Ex 12-2: malloc and assign
-------------after malloc-------------
   9925952 (000000000061FE10)
--------------------------------------
index |     value    | memory location
--------------------------------------
    0 |         10 | 0000000000971550
    1 |         11 | 0000000000971554
    2 |         12 | 0000000000971558
    3 |         13 | 000000000097155C
    4 |         14 | 0000000000971560
-------------after assign-------------
        10 (000000000061FE10)
```

malloc

void *malloc(size_t size)

Lab 12-1:

上一次的課程中，我們有提到可以利用memset()，指標變數的特定範圍內，全部變成同一個特定字元。在前一個範例EX12-2中，利用for loop做改0的動作十分沒有效率，如果利用memset就可以一次改完全部element內的數值，而且還省去一個for loop。請利用memset()將malloc所配置的記憶體空間都填上0。

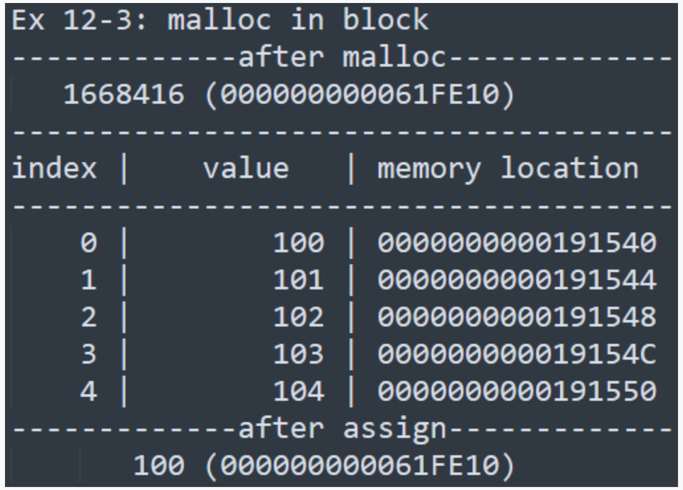


```
--------------------------------------
index |     value    | memory location
--------------------------------------
    0 |            0 | 00000000001B1550
    1 |            0 | 00000000001B1554
    2 |            0 | 00000000001B1558
    3 |            0 | 00000000001B155C
    4 |            0 | 00000000001B1560
```

malloc in block

```
void *malloc(size_t size)
```

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-3: malloc in block*/
        printf("Ex 12-3: malloc in block\n");
        int size = 5, i;
        if(1){ // if else block
                int *p = (int*) malloc(sizeof(int)*size);
                printf("%10d (%p)\n", *p, &p);
                for (i=0; i<size; i++){
                        p[i] = i+100;
                        printf("%5d | %10d | %p\n", i, p[i], &p[i]);
                }
                printf("%10d (%p)\n", *p, &p);
        }
        // printf("%10d (%p)\n", *p, &p); // error: 'p' undeclared (first use in this function)
}
```



```
Ex 12-3: malloc in block
------------after malloc------------
   1668416 (000000000061FE10)
------------------------------------
index |     value     | memory location
------------------------------------
    0 |           100 | 0000000000191540
    1 |           101 | 0000000000191544
    2 |           102 | 0000000000191548
    3 |           103 | 000000000019154C
    4 |           104 | 0000000000191550
------------after assign------------
       100 (000000000061FE10)
```

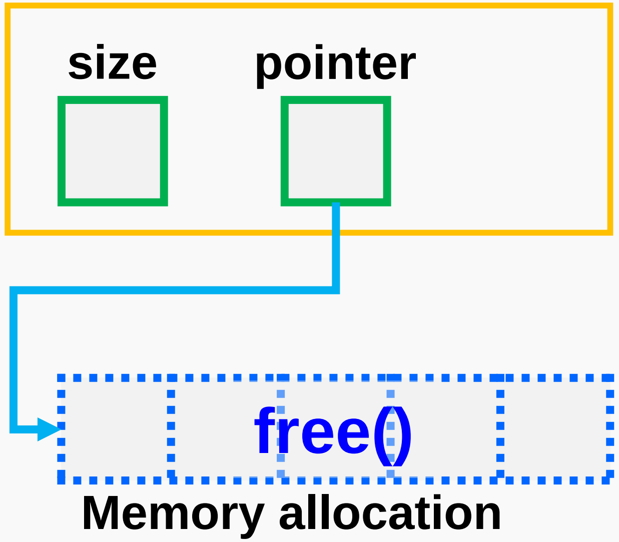
Block內配置的記憶體：離開block的時候，內部的變數就會被釋放。但是記憶體的數值並沒有reset。

free memory space

一旦做了動態記憶體配置，就一定要記得釋放掉!

釋放的方法就是使用free()函數。

```
void free(void *ptr)
```

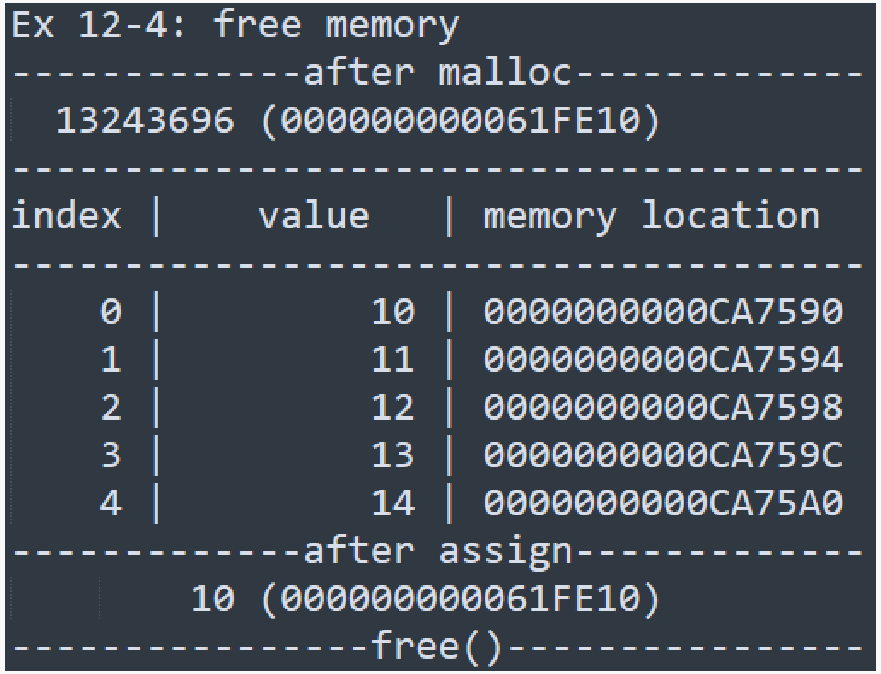


不然就可能會造成記憶體流失(memory leak)的問題。一般來說，釋放記憶體有幾個好處，想像現在你要儲存一組信用卡資料(號碼、安全碼、姓名)，交易完需要刪除資料，以免被別人盜取，此時就可以用動態記憶體配置的作法，使用完回收記憶體。但是這樣可能還是不夠。。。為甚麼呢???

free memory space

void free(void *ptr)

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-4: free memory*/
        printf("Ex 12-4: free memory\n");
        int size = 5, i;
        int *p = (int*) malloc(sizeof(int)*size);
        printf("%10d (%p)\n", *p, &p);
        for (i=0; i<size; i++){
                p[i] = i+10;
                printf("%5d | %10d | %p\n", i, p[i], &p[i]);
        }
        printf("%10d (%p)\n", *p, &p); // after assign
        free(p); // free memory
}
```

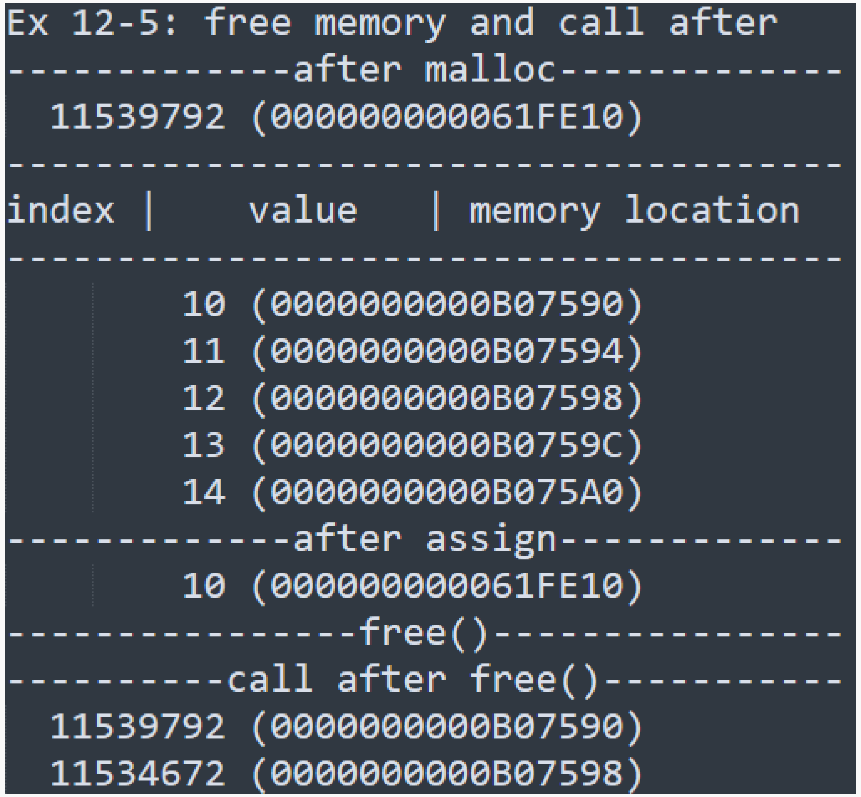


```
Ex 12-4: free memory
-------------after malloc-------------
  13243696 (000000000061FE10)
--------------------------------------
index |     value    | memory location
--------------------------------------
    0 |           10 | 0000000000CA7590
    1 |           11 | 0000000000CA7594
    2 |           12 | 0000000000CA7598
    3 |           13 | 0000000000CA759C
    4 |           14 | 0000000000CA75A0
-------------after assign-------------
        10 (000000000061FE10)
----------------free()----------------
```

free memory and call again

void free(void *ptr)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-5: free memory and call after*/
        printf("Ex 12-5: free memory and call after\n");
        int size = 5, i;
        int *p = (int*) malloc(sizeof(int)*size);
        printf("%10d (%p)\n", *p, &p);
        for (i=0; i<size; i++){
                p[i] = i+10;
                printf("%5d | %10d | %p\n", i, p[i], &p[i]);
        }
        printf("%10d (%p)\n", *p, &p); // after assign
        free(p); // free memory
        printf("%10d (%p)\n", p[0], &p[0]);
        printf("%10d (%p)\n", p[2], &p[2]);
}
```

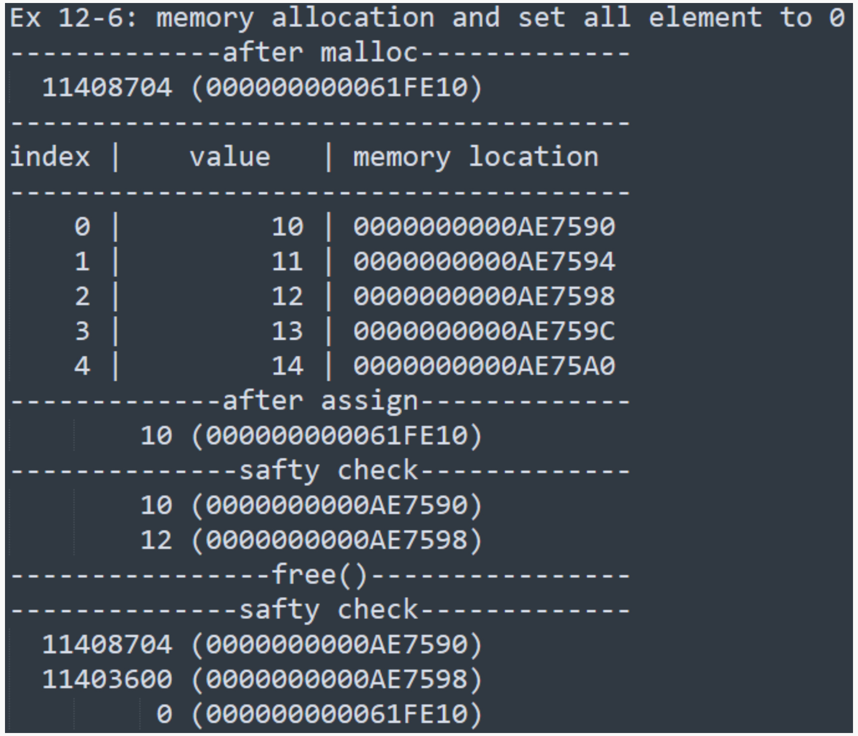


```
Ex 12-5: free memory and call after
-------------after malloc------------
  11539792 (000000000061FE10)
-------------------------------------
index |     value    | memory location
-------------------------------------
        10 (0000000000B07590)
        11 (0000000000B07594)
        12 (0000000000B07598)
        13 (0000000000B0759C)
        14 (0000000000B075A0)
-------------after assign------------
        10 (000000000061FE10)
---------------free()----------------
----------call after free()----------
  11539792 (0000000000B07590)
  11534672 (0000000000B07598)
```

Free & Set to 0

void free(void *ptr)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-6: memory allocation and set all element to 0*/
        printf("Ex 12-6: memory allocation and set all element to 0\n");
        int size = 5, i;
        int *p = (int*) malloc(sizeof(int)*size);
        printf("%10d (%p)\n", *p, &p);
        for (i=0; i<size; i++){
                p[i] = i+10;
                printf("%5d | %10d | %p\n", i, p[i], &p[i]);
        }
        printf("%10d (%p)\n", *p, &p); // after assign
        free(p); // free memory
        printf("%10d (%p)\n", p[0], &p[0]);
        printf("%10d (%p)\n", p[2], &p[2]);
        p = 0;
        // printf("%10d (%p)\n", p[0], &p[0]); // cannot use anymore
        printf("%10d (%p)\n", p, &p);
}
```

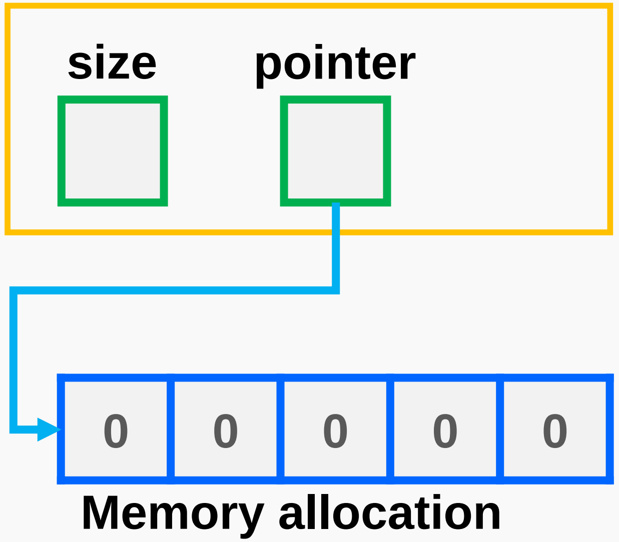


```
Ex 12-6: memory allocation and set all element to 0
-------------after malloc------------
  11408704 (000000000061FE10)
-------------------------------------
index |     value    | memory location
-------------------------------------
    0 |         10 | 0000000000AE7590
    1 |         11 | 0000000000AE7594
    2 |         12 | 0000000000AE7598
    3 |         13 | 0000000000AE759C
    4 |         14 | 0000000000AE75A0
-------------after assign------------
        10 (000000000061FE10)
--------------safty check------------
        10 (0000000000AE7590)
        12 (0000000000AE7598)
----------------free()---------------
--------------safty check------------
  11408704 (0000000000AE7590)
  11403600 (0000000000AE7598)
         0 (000000000061FE10)
```

calloc

在前面的範例中，會不會覺得用malloc配置記憶體完還需要再寫一個程式，將數值設為0，不覺得很麻煩嗎? 這個時候你就可以用calloc函數，它會自動將數值設為0。

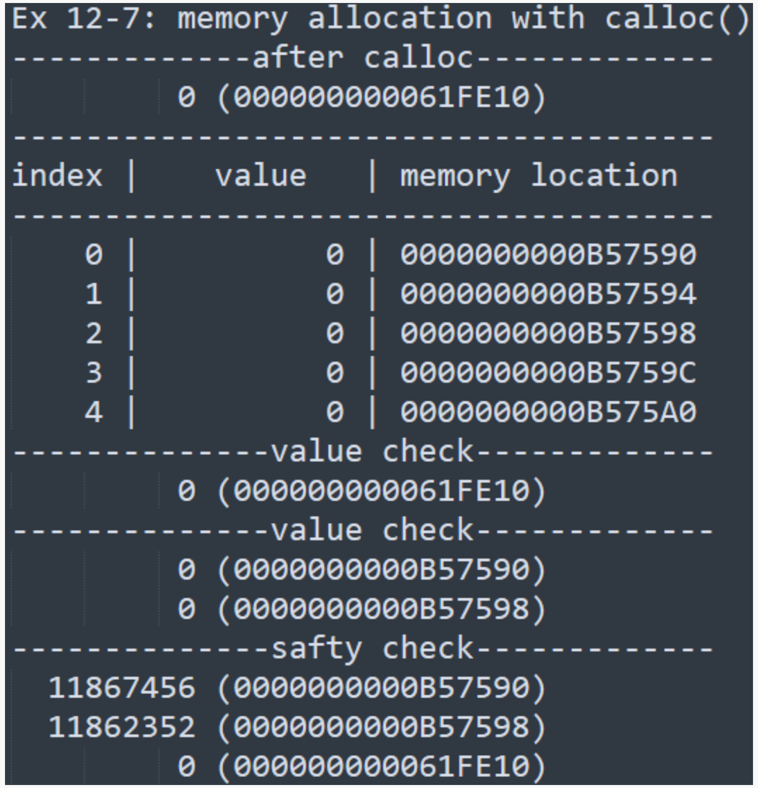
```
void *calloc(size_t nitems, size_t size)
```



calloc

```
void *calloc(size_t nitems, size_t size)
```

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-7: memory allocation with calloc()*/
        printf("Ex 12-7: memory allocation with calloc()\n");
        int size = 5, i;
        int *p = (int*) calloc(size, sizeof(int));
        printf("%10d (%p)\n", *p, &p);
        for (i=0; i<size; i++){
                printf("%5d | %10d | %p\n", i, p[i], &p[i]);
        }
        printf("%10d (%p)\n", *p, &p); // after assign
        printf("%10d (%p)\n", p[0], &p[0]);
        printf("%10d (%p)\n", p[2], &p[2]);
        free(p); // free memory
        printf("%10d (%p)\n", p[0], &p[0]);
        printf("%10d (%p)\n", p[2], &p[2]);
        p = 0;
        printf("%10d (%p)\n", p, &p);
}
```

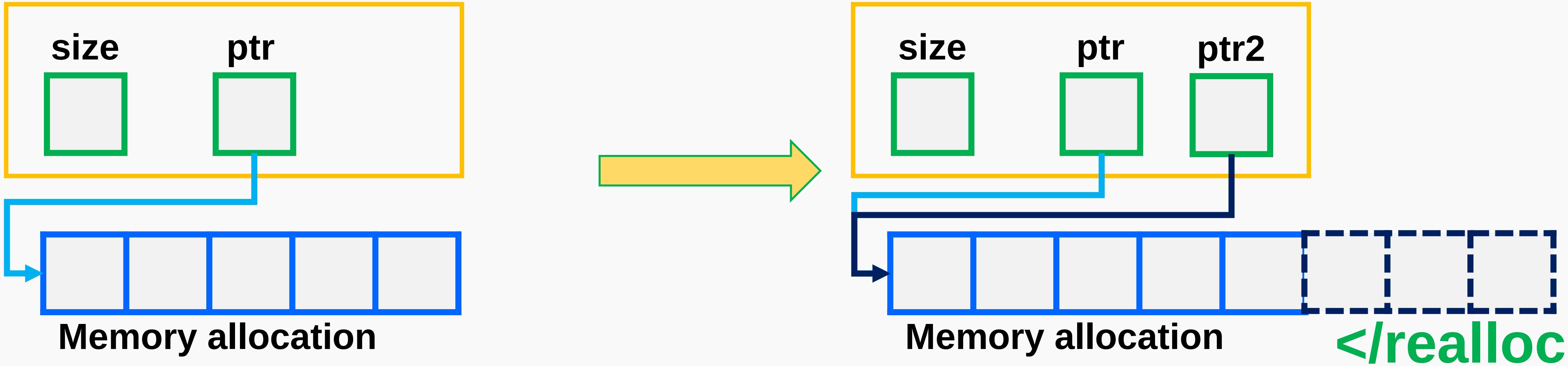


```
Ex 12-7: memory allocation with calloc()
-------------after calloc-------------
         0 (000000000061FE10)
--------------------------------------
index |    value    | memory location
--------------------------------------
    0 |          0 | 0000000000B57590
    1 |          0 | 0000000000B57594
    2 |          0 | 0000000000B57598
    3 |          0 | 0000000000B5759C
    4 |          0 | 0000000000B575A0
-------------value check-------------
         0 (000000000061FE10)
-------------value check-------------
         0 (0000000000B57590)
         0 (0000000000B57598)
-------------safty check-------------
  11867456 (0000000000B57590)
  11862352 (0000000000B57598)
         0 (000000000061FE10)
```

Dynamic 1D Array - realloc

有時候我們會面臨到，已經配置好的記憶體空間需要被調整大小，如果我們需要更多的記憶體位置的時候，就可以使用到realloc函數。這個函數主要目的，就是再跟系統要多的記憶體空間配置到指定的pointer。

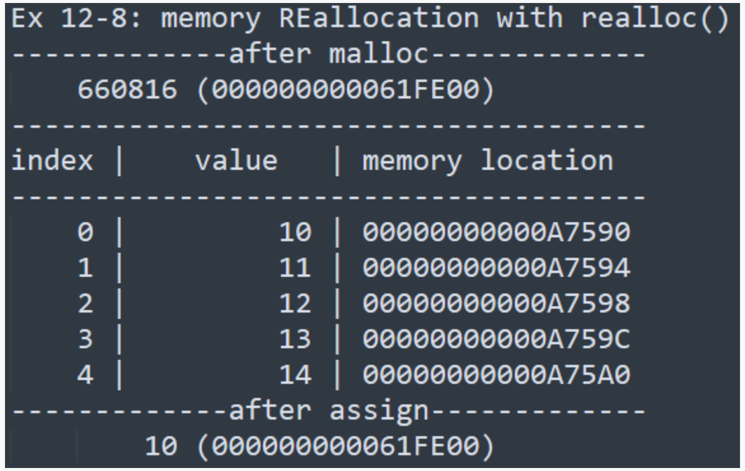
```
void *realloc(void *ptr, size_t size)
```



<realloc/>

void *realloc(void *ptr, size_t size)

Dynamic 1D Array - realloc



```
Ex 12-8: memory REallocation with realloc()
-------------after malloc-------------
    660816 (000000000061FE00)
--------------------------------------
index |    value    | memory location
--------------------------------------
    0 |          10 | 00000000000A7590
    1 |          11 | 00000000000A7594
    2 |          12 | 00000000000A7598
    3 |          13 | 00000000000A759C
    4 |          14 | 00000000000A75A0
-------------after assign-------------
        10 (000000000061FE00)
```

```
#include <stdio.h>
#include <stdlib.h>
int main(){
        /*Ex 12-8: memory REallocation with realloc()
        printf("Ex 12-8: memory REallocation with realloc()\n");
        int size = 5, i;
        int *arr1 = (int*) malloc(sizeof(int)*size);
        printf(“%10d (%p)\n”, *arr1, &arr1);
        for (i=0; i<size; i++){
                arr1[i] = i + 10;
                printf("%5d | %10d | %p\n", i, arr1[i], &arr1[i]);
        }
        printf("%10d (%p)\n", *arr1, &arr1); // after assign
}
```

</realloc>

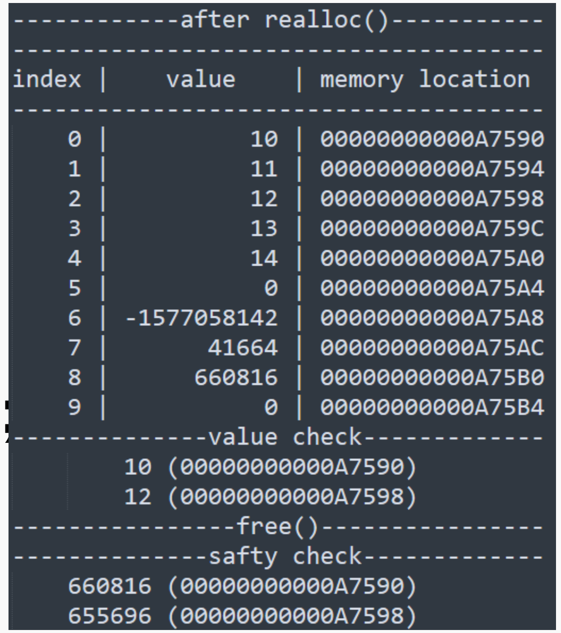
<realloc/>

void *realloc(void *ptr, size_t size)

Dynamic 1D Array - realloc

```
int *arr2 = realloc(arr1, sizeof(int)*size*2);
printf(“%10d (%p)\n”, *arr1, &arr1);
for (i=0; i<size*2; i++){
        printf("%5d | %10d | %p\n", i, arr2[i], &arr2[i]);
}
printf("%10d (%p)\n", arr1[0], &arr1[0]);
printf("%10d (%p)\n", arr1[2], &arr1[2]);

// free(arr1); <= that is unnecessary
free(arr2); // safe and okay
printf("%10d (%p)\n", arr1[0], &arr1[0]);
printf("%10d (%p)\n", arr1[2], &arr1[2]);
```



```
------------after realloc()-----------
--------------------------------------
index |     value     | memory location
--------------------------------------
    0 |            10 | 00000000000A7590
    1 |            11 | 00000000000A7594
    2 |            12 | 00000000000A7598
    3 |            13 | 00000000000A759C
    4 |            14 | 00000000000A75A0
    5 |             0 | 00000000000A75A4
    6 |   -1577058142 | 00000000000A75A8
    7 |         41664 | 00000000000A75AC
    8 |        660816 | 00000000000A75B0
    9 |             0 | 00000000000A75B4
-------------value check--------------
        10 (00000000000A7590)
        12 (00000000000A7598)
---------------free()-----------------
-------------safty check--------------
    660816 (00000000000A7590)
    655696 (00000000000A7598)
```

```
}
```

</realloc>

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