國立屏東大學 資訊工程學系 程式設計(二)

17. 記憶體配置與管理

- 變數的範疇與生命週期
  - 自動變數
  - 區域變數/區塊變數
  - 全域變數
  - 靜態變數
- 動態記憶體配置
- C語言程式的記憶體佈局

在第12章的12.4節，我們首次介紹了區域變數與全域變數的概念，本章將進一步說明變數的生命週期與可視範圍，最後並將介紹C語言的程式的記憶體佈局。

-範疇(scope)

變數的範疇(scope)又稱為可視性(visibility)或稱為可作用範圍，是指變數可有效使用的範圍，可分為三類：

- 區域變數(local variables)宣告在某個函式(包含main()函式)內的變數，其作用範圍僅限於其所在的函式內。
- 區塊變數(block variables)宣告在某個函式內的程式區塊內的變數，其作用範圍僅限於其所在的區塊內。
- 全域變數(global variables)宣告在函式之外(意即其宣告不位於任何的函式內)，可在程式任何地方使用。

上述所謂的區塊是指在程式碼中，以「{」開始至「}」為止的程式碼段落，請參考下面的例子：

```
#include <stdio.h>

int g; // global variable

void foo(int a, int b)
{
   int x=1, y=2, z;    //  local variables: a, b, x, y, z
   z = a+b;
   printf("x=%d y=%d z=%d g=%d \n", x, y, z, g); // x=1 y=2 z=15 g=8;
}

int main()
{
   int x=3, y=5;
   g = x+y;
   printf("x=%d y=%d g=%d\n", x, y, g); // x=3 y=5 g=8;
```

```
    {
        int z;
        z = x+y;
        printf("x=%d y=%d z=%d g=%d\n", x, y, z, g); // x=3 y=5 z=8 g=8;
    }

    foo(6, 9);

}
```

-生命週期(lifetime)

變數的生命週期是指其存在於記憶體內的時間，上一節所提到的三種變數，其生命週期如下：

- 區域變數(local variables)：從其宣告開始至函式結束為止。
- 區塊變數(block variables)：從其宣告開始至區塊結束為止。
- 全域變數(global variables)：從程式開始到程式結束為止。

在程式中的變數，其實只是一個符號，用以代表某個值(value)：所謂的程式設計，就是透過程式碼對這些值進行邏輯上的操作，以滿足特定的應用目的。在程式執行時，變數所代表的值，必須存在於記憶體內，才能進行各式操作。由於記憶體是有限的，所以變數所對應的記憶體空間，也需要有妥善的方法管理：C語言在這方面可分成三種處理方法：自動(automatic)：靜態(static)與動態(dynamic)：

17.0.1 自動記憶體配置

在C語言中，區域變數與區塊變數是以自動的方式管理。每當變數被宣告後，就會自動地在記憶體中配置適合的空間供其使用；當變數所在的函式或區塊結束時，所配置的記憶體空間就會被釋放。

從這個角度來看，宣告在main()函式內的變數，其記憶體空間是從其宣告開始進行配置，一直到main()函式結束(也就是程式結束時)，才會被釋放。對於程式中存在的其它函式而言，函式內所宣告的變數的記憶體空間，也是從宣告開始進行配置，但其所在的函式結束(或返回時)，就會被釋放。在宣告時，如有給定初始值，則依其值填入記憶體內，否則保留該記憶體原有的值(通常是對程式而言無意義的資料，最好要記得將變數的初始值明確地加以設定)。

以這種方式使用記憶體的變數又稱為自動變數，每次在其所處的函式或區塊內。

* 一般是我們放在main() function()幫忙運算，或是for(int i = 0)所用到的變數。 * 有效範圍是從被宣告開始，到區塊的結束。 * 一開始不設定值的話，內容值會是堆疊中沒有用的資料，為一無用的值。 * 每次區塊重新執行會重新建立此變數，若有初值，每次建立會重新指定初值。 *此資料存在堆疊之中，非資料段，故使用完，超過block會將裡面的資料清除。

17.0.2 靜態記憶體配置

在程式編譯時，全域變數與字串常值(stringliteral)就會被配置到一塊記憶體空間，且在程式執行的過程中，所配置的空間將持續保留給這些變數使用，直到程式結束為止，我們將此種方式稱為靜態記憶體配置。除了全域變數與字串常值外：C語言允許我們在變數宣告時，使用：static：來修飾此宣告，將該變數的記憶體空間強制以靜態方式處理。例如：

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

```
void foo()
{
  static int i=1;
  printf("i=%d\n", i++);
}

int main()
{
  int i;
  for(i=0;i<10;i++)
    foo();

  return 0;
}
```

在此例中，foo()函式內的變數i，被宣告為，static，，其結果會在編譯時就配置好所需的記憶體空間，且其生命週期亦延長至程式結束為止，我們將其稱為「靜態變數」。要注意的是，在foo()函式內的，static int i=1;，宣告，其中，i=1，是初始的設定，只會作用一次，當foo()函式再次(及後續每一次)被呼叫時，將不會再設定其初始值。如果沒有設定初始值的話，全域變數與靜態變數將會以0做為其預設的初始值。

17.0.3 動態記憶體配置

動態記憶體配置，是由我們明確地以，malloc，等指令來取得記憶體空間，並以，free，釋放不再需要的記憶體空間。以此種方式配置的記憶體空間是不會自動被釋放的，如果我們沒有在程式中使用，free，來釋放，則其生命週期將一直持續到程式結束為止。通常都是以指標來存取這些動態配置的記憶體空間，我們必須小心的以指標來操作這些指向動態配置的記憶體空間，假設在程式中，沒有任何指標指向一個動態配置的記憶體空間，那麼該空間將無法被使用也無法被釋放，我們將此現像稱為記憶體洩漏(memory leak)，

C語言提供以下三個有關動態記憶體配置的函式，它們的函式原型定義於，stdlib.h，中：

- void * malloc(unsigned int size)，配置一塊大小為size的記憶體空間，但不進行初始化。
- void * calloc(unsigned int nelem, unsigned int elsize)，配置nelem個元素，其中每個元素的大小為elsize，與malloc不同的是，所配置到的空間的內容會被清空，意即其初始值會被設定為0。
- void * realloc(void *prt, unsigned int newSize)，改變由ptr指標所指向的記憶體空間的大小，將其改成newSize的大小。

這三個函式的傳回值都一樣，當配置成功時傳回其所配置的空間的記憶體位址，這是以，void *，為型態的傳回值，，void *，代表指標，但不指定其參考型態，換言之，所傳回的記憶體位址內所儲存的可以是任意型態的資料。我們將，void *，稱為是「泛型指標(generic pointer)，，當記憶體空間不足或其它原因無法成功地配置記憶體時，則傳回NULL，NULL被定義在多個函式標頭檔中，例如locale.h，stddef.h，stdio.h，stdlib.h，string.h及time.h中，代表「空」、「無」等狀態，在動態記憶體配置方面，，NULL，代表的是「空指標(pointer to nothing)，，也就是沒有指向任何地方的指標。在C語言的實作上，，NULL，是以數值0加以定義。

以下的程式動態配置了可以存放1000個整數的記憶體空間(假設一個整數為32位元)：

```
int *p;

p = malloc(4000); //配置1000個整數
if(p==NULL)
  printf("Allocation failed!\n");


或是

if((p=malloc(1000*sizeof(int))) == NULL)
  printf("Allocation failed!\n");


或是

if((p=calloc(1000, 4))==NULL)
   printf("Allocation failed!\n");

或是

if((p=calloc(1000, sizeof(int)))==NULL)
   printf("Allocation failed!\n");

//我們也可以動態地改變指標p所指向的記憶體空間的大小，例如：

if((p = realloc(p, 2000*sizeof(int)))==NULL)
   printf("Resize failed!\n");
```

這些所配置到的空間，當不再使用時，必須以free來加以釋放：

```
free(p);
```

17.1 C語言程式的記憶體佈局

典型的C語言程式經編譯後，其記憶體配置具有以下六個區段，如figure 1所示：

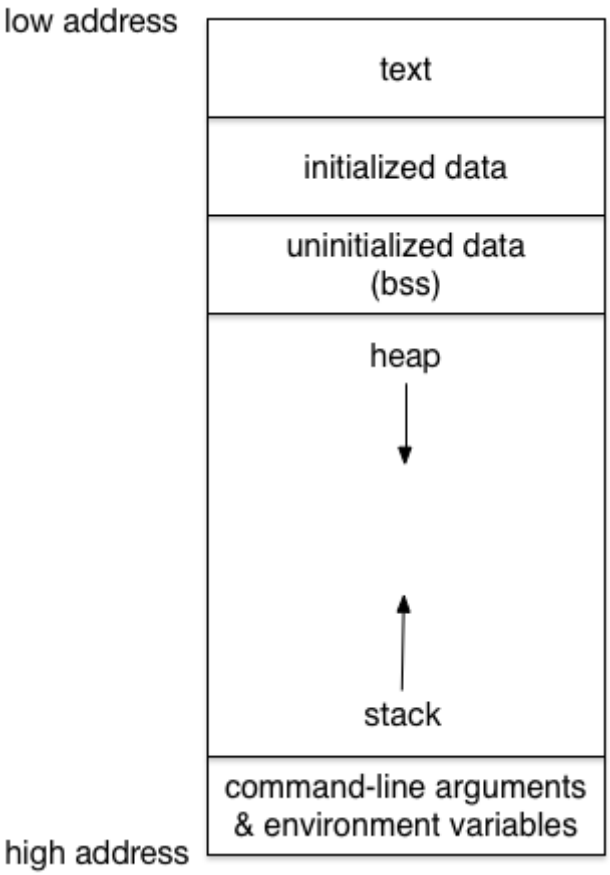

Fig. 1: 典型的C語言記憶體佈局

1. 文字區段(Text segment):又稱為程式碼區段(code segment)，用以存放編譯後所產生的機器碼(machine code)。
2. 初始化資料區段(Initialized data segment):有時也簡稱為資料區段(data segment)，用以存放在程式中有給定不為0的初始值的全域變數、靜態變數與字串常值。
3. 未初始化資料區段(Uninitialized data segment):又常被稱為bas區段(block started by symbol)，用以存放未宣告初始值或宣告為0的全域變數與靜態變數。未給定初始值的全域變數與靜態變數，在程式開始執行前，其數值會被設定為0。
4. 命令列引數與環境變數(command-line arguments and environment variables):放置使用者執行程式時所傳入的命令列引數與相關環境變數。
5. 堆疊(Stack): 放置區域變數與區塊變數(也稱為自動變數)的地方，以及每次函式呼叫時，儲存資訊的地方(包含函式未來返回的位址以及呼叫當時的處理器暫存器值等)。一個函式一旦被呼叫執行，其所需的區域或區塊變數也會在Stack中被配置，Stack中的資料是以後進先出(Last-in First-out，LIFO)的方式管理，先配置的變數會放在底層，最後配置的變數放在最上層。<課堂討論>
6. 堆積(Heap): 此區段是用以放置動態配置的記憶體的空間，與Stack不同的是，Heap與Stack成長的方向相反，但與Stack區段共用同一塊記憶體空間。在此區段所使用的空間，從配置開始直到使用，free，釋放以前都會存在。