(\*ptr)++ 會直接針對內容+1

\*ptr++ = \*(ptr++) = 下一格ptr的內容

^: XOR

~: 補數

程式執行過程 分: globa(permanent), stack, heap

Stack: 區域變數 含式參數 函數的返回位址

void method2() {

  method1();

}

上述當呼叫method1()時，系統會先把method2的返回位址存到stack當中，為何是存放在stack呢，因為函式的呼叫有後進先出的概念，當method1()被呼叫而開始執行，待結束時必定會查找該返回何處，故最後一定會讀取函式的返回位址，既然如此明確而有條理，當然也是往stack放！

**可預測性外加後進先出的生存模式，令stack無疑是最佳的存放策略**

**這些不可預測的因素造成上述的stack區塊不適合運用於此。當資訊為動態配置產生，系統會存放在另外一塊空間，稱之為『Heap』**

在C++或Java中利用new語法產生的就是動態配置的物件，需存放於heap中

Heap中的資料如果沒有正常的回收，將會逐步成長到將記憶體消耗殆盡

stack overflow一般是因為過多的函式呼叫(例如：遞迴太深)、或區域變數使用太多，此時請試著將stack size調大一點，另外檢查看看函式的呼叫跟變數的使用量

heap overflow請檢查是否都有正確將heap space的資料回收，另外採行的動態配置是否合理，不要過渡濫用而new出無謂的空間，若真的是程式過於複雜造成，請將heap size調大一些。

當變數有宣告時加上static限定時，一但變數生成，它就會一直存在記憶體之中，即使函式執行完畢，變數也不會消失

extern可以聲明變數會在其它的位置被定義，這個位置可能是在同一份文件之中，或是在其它文件之中，例如： 

* some.c

**double** someVar = 1000;  
// 其它定義 ...

* main.c

#include <stdio.h>  
  
**int** main(**void**) {  
 **extern** **double** someVar;  
   
 printf("%f\\n", someVar);  
   
 **return** 0;  
}

在main.c中實際上並沒有宣告someVar，extern指出someVar是在其它位置被定義，編譯器會試圖在其它位置或文件中找出 someVar的定義，結果在some.c中找到，因而會顯示結果為1000，要注意的是，extern聲明someVar在其它位置被定義，如果您 在使用extern時同時指定其值，則視為在該位置定義變數，結果就引發重覆定義錯誤，例如：

**#include <stdio.h>  
  
int main(void) {  
    extern double someVar = 100; // error, `someVar' has both `extern' and initializer       
    ...  
    return 0;  
}**

您必須先聲明extern找到變數，再重新指定其值，這麼使用才是正確的：

**#include <stdio.h>**  
  
**int main(void) {**  
**extern double someVar;  
    someVar = 100;  
    ...**  
**return 0;**  
**}**