Strong Symbol and Weak Symbol(強型別和弱型別)

Compiler 會把已經初始化的 Global varible 當作 Strong Symbol
未初始化的 Global varible 為 Weak Symbol
我們可以使用 GCC 提供的 "__attribute__((weak))"
來定義任意一個 Strong Symbol 為 Weak Symbol
以下是一個例子:

```
1. extern int ext;
2.
3. int weak;
4. int strong = 1;
5. __attribute__((weak)) weak2 = 2;
6.
7. int main(){
8.     return 0;
9. }
```

在這個例子中 weak 和 weak2 都是 Weak Symbol
strong 和 main 是 Strong Symbol
ext 不是 Strong 也不是 Weak，因為他是一個外部變數

Compiler 會按照下列規則處理 Strong 以及 Weak Symbol

Rule1: Strong Symbol 不能在不同的 Obj 檔被多次定義
這就是我們常常看到的重複定義錯誤

Rule2: 如果一個 Symbol 在某個檔案是 Strong，在其他檔案都是 Weak
Compiler 會選擇 Strong Symbol

Rule3: 如果一個 Symbol 在每個檔案都是 Weak，會選擇最大的 Type
比如說一個同樣名稱的 int 和 double global varible
Compiler 在 Link 的時候會選擇 double

外部符號的 Reference 也有分兩種

Strong Reference: 在 Link 時找不到符號定義會回報錯誤
Weak Reference: 在 Link 時找不到符號定義不會回報錯誤，通常會預設為 0

下面是 GCC 把 foo()宣告成 weak reference 的擴充 keyword

```
1. __attribute__((weakref)) void foo();
2.
3. int main()
4. {
5.     foo();
6. }
```

上面這段 code 可以編譯成執行檔且不會產生錯誤

但是我們執行程式的話，因為沒有定義 foo()，foo 的位置為 0
因此會發生不合法的位置存取錯誤

Weak Symbol 和 Weak Reference 對函式庫的設計非常有用
函式庫可以定義一些 Weak Symbol 的函式
使用者可以自己定義一些 Strong Symbol 達到擴充功能
因為 Strong Symbol 會蓋掉 Weak Symobl

我們也可以透過 Weak Reference 使用一些擴充功能
以後就算我們把擴充功能去掉，程式還是可以正常 Link