# Other Advanced Topic on DLL

井民全製作
http://debut.cis.nctu.edu.tw/~ching

# 使用不同的 Export name

MyDLL.def 產生

__declspec(dllexport) 產生

- 你可以在 DLL 的 **.def** 檔中加入其他的 export name

1

```
LIBRARY        MyDLL
EXPORTS
Addnew=Add
```

MyDLL.def

使用 Addnew
也可以呼叫 Add

2 RVA 是一樣的

```
ordinal hint RVA      name

      1    0 0001785C Add
      2    1 0001785C Addnew
      3    2 00051440 g_nResult
```

dumpbin /exports MyDLL.dll

注意: 這裡我們使用 Addnew 呼叫 function

3

```
typedef int (*ADDPF)(int,int);
int _tmain(int argc, _TCHAR* argv[])
{
        HMODULE hModule=LoadLibrary("MyDLL.dll");
        ADDPF pf=(ADDPF)GetProcAddress(hModule,"Addnew");
        int c=(*pf)(1,2);
        return 0;
}
```

# 加入 ordinal number

```
LIBRARY        MyDLL
EXPORTS
Addnew=Add @12345
```

MyDLL.def

指定 Addnew 的 ordinal number= 12345

沒有加 @12345 之前

```
ordinal hint RVA      name

      1    0 0001785C Add
      2    1 0001785C Addnew
      3    2 00051440 g_nResult
```

dumpbin /exports MyDLL.dll

DLL 的部分

```
ordinal hint RVA      name

  12346    0 0001785C Add
  12345    1 0001785C Addnew
  12347    2 00051440 g_nResult
```

__declspec(dllexport) 產生

MyDLL.def 產生

```
ordinal    name

           _Add
           _Addnew
           _g_nResult
```

dumpbin /exports MyDLL.lib

lib 的部分

```
ordinal    name

  12345    _Add
           _Addnew
           _g_nResult
```

# 隱藏你 exported 的 name

- 在 .def 中加入
  - NONAME 表示
    - 使用者需要使用 ordinal 呼叫你的 function
    - export name 不會放在 .dll 檔中, 可降低 dll的大小
  - PRIVATE 表示 .def 中 EXPORTS 的 name 不會放在 .lib 中

# 建立方法

**Step 1**

先把 __declspec(dllexport) 拿掉

```
#ifdef MYLIBExport
   #define  MYLIBAPI extern "C" //__declspec(dllexport)
#else
   #define  MYLIBAPI extern "C" //__declspec(dllimport)
#endif
MYLIBAPI int g_nResult;
MYLIBAPI int Add(int nLeft, int nRight);
```

MyDLL.h

**Step 2**

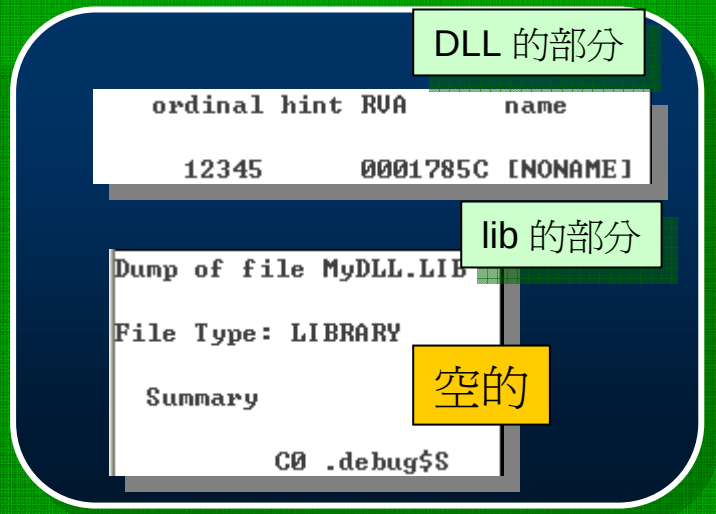
加入 NONAME 與 PRIVATE 保留字

```
LIBRARY          MyDLL
EXPORTS
Addnew=Add @12345 NONAME
```

MyDLL.def

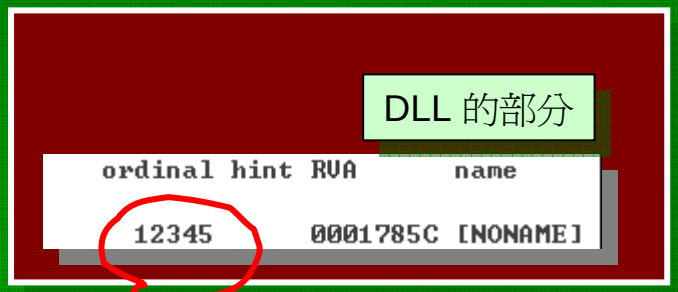
**Step 3**

檢查一下 (dumpbin)



# 沒有 name 怎麼呼叫?

- 使用 ordinal number




```
#include <windows.h>
typedef int (*ADDPF)(int,int);
int _tmain(int argc, _TCHAR* argv[])
{
        HMODULE hModule=LoadLibrary("MyDLL.dll");
        ADDPF pf=(ADDPF)GetProcAddress(hModule,(LPCSTR)12345);
        int c=(*pf)(1,2);
        return 0;
}
```

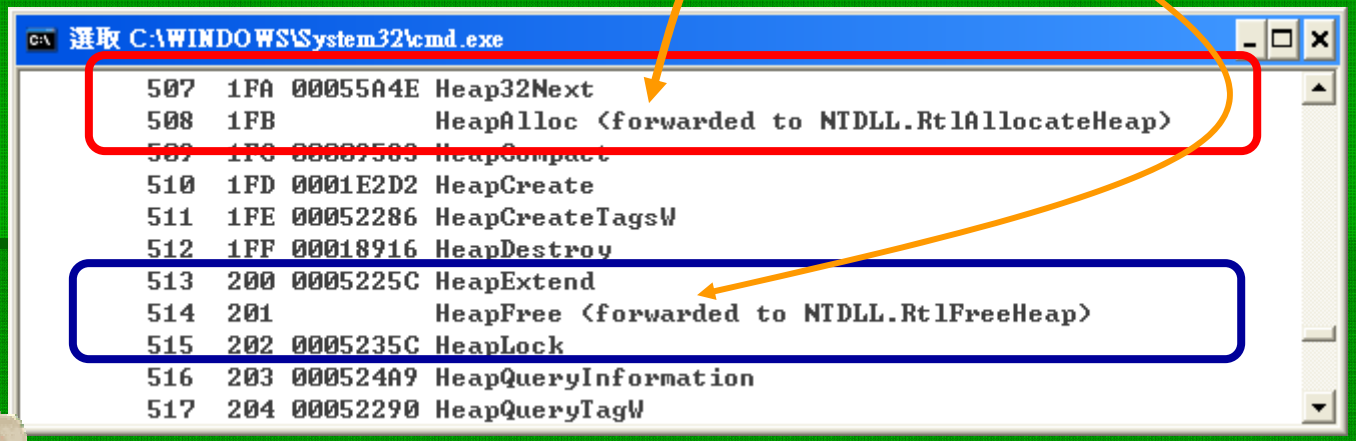
使用 ordinal number 取出 Function 所在的位址

完成呼叫的動作

# DLL Forward

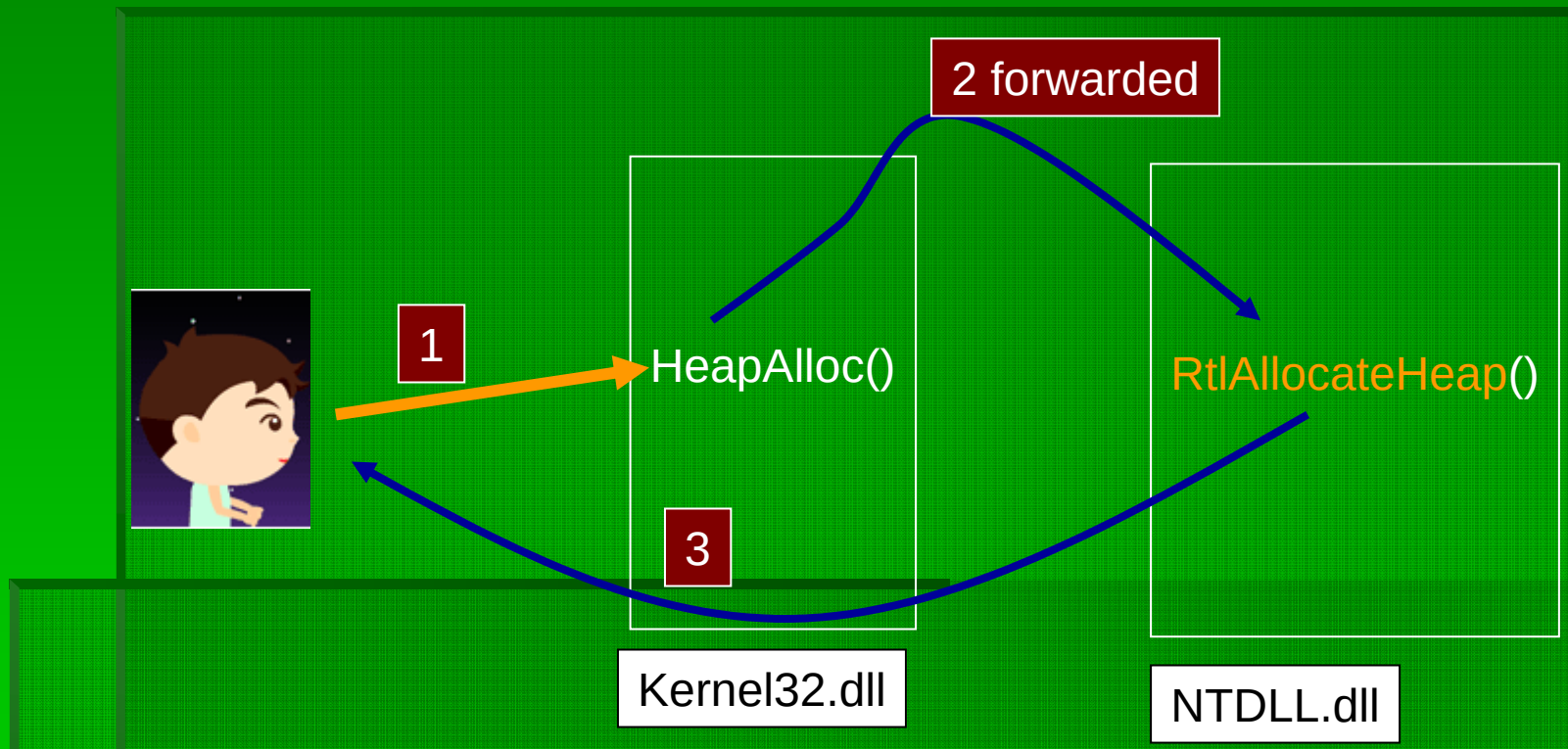
- 若你使用 DumpBin 看 Kernel32.dll 你會發現

看到了 forwarded 字樣了嗎?




意思是當你的程式呼叫 HeapAlloc ,
實際上呼叫的是 RtlAllocateHeap

# DLL Forward


2 forwarded
1
HeapAlloc()
RtlAllocateHeap()
3
Kernel32.dll
NTDLL.dll

# 如何讓我也能操作 forwarded?

很簡單, 只要加入

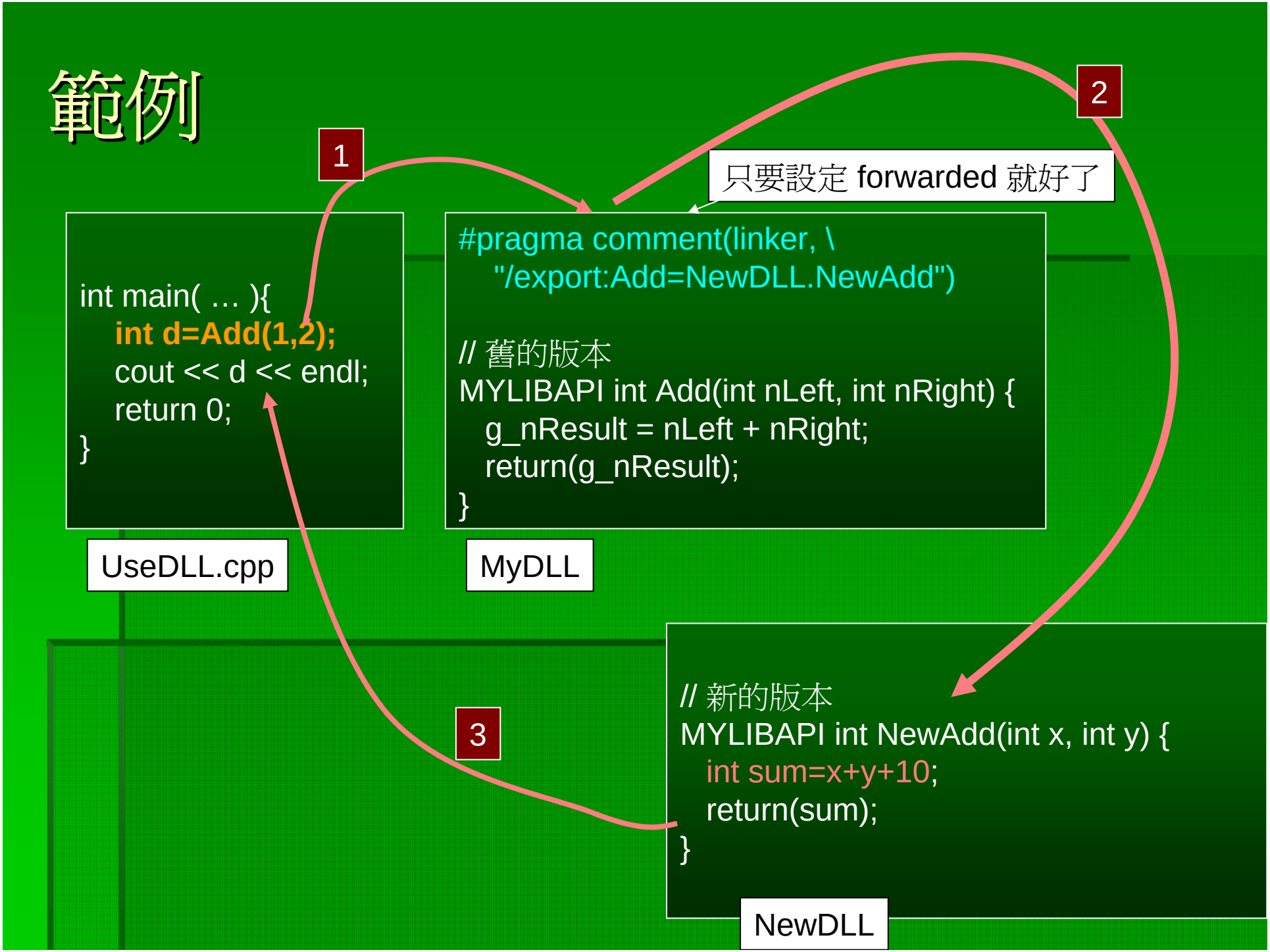
原先的 name

真正執行的 function

```
#pragma comment(linker, "/export:Add=NewDLL.NewAdd")
```

新DLL name

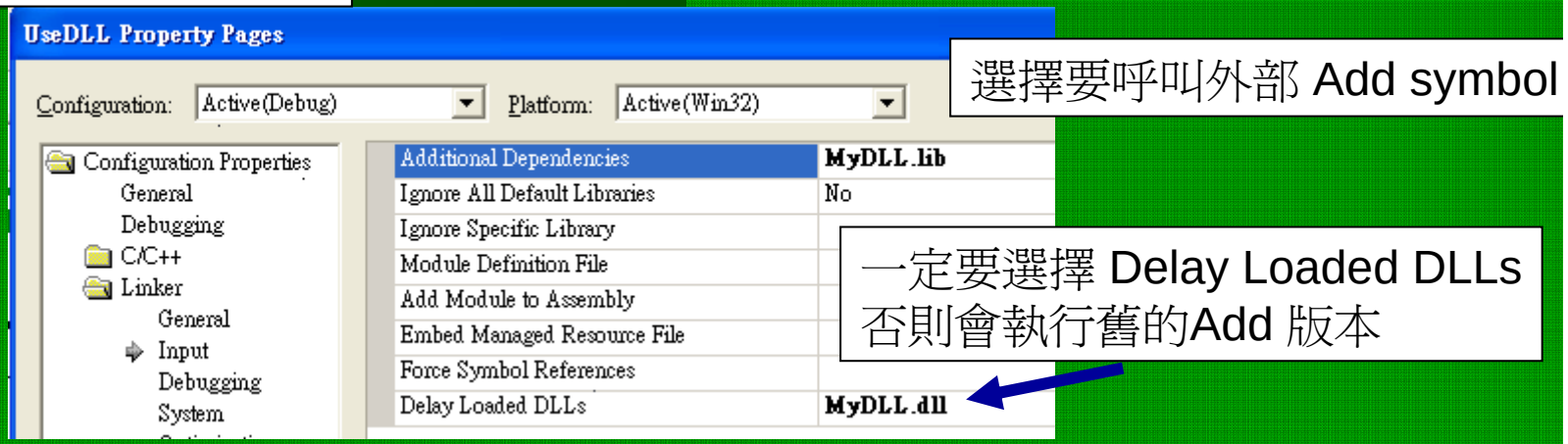
# 範例


1
2
3
只要設定 forwarded 就好了
int main( … ){
int d=Add(1,2);
cout << d << endl;
return 0;
}
UseDLL.cpp
#pragma comment(linker, \
"/export:Add=NewDLL.NewAdd")
// 舊的版本
MYLIBAPI int Add(int nLeft, int nRight) {
g_nResult = nLeft + nRight;
return(g_nResult);
}
MyDLL
// 新的版本
MYLIBAPI int NewAdd(int x, int y) {
int sum=x+y+10;
return(sum);
}
NewDLL

# 測試

1. 編譯 NewDLL 產生 NewDLL.dll 與 NewDLL.lib
2. 編譯 MyDLL 產生 MyDLL.dll 與 MyDLL.lib
3. 編譯 UseDLL
   - 先把 MyDLL.lib 放在可搜尋到的地方
   - 執行時, NewDLL.dll 與 MyDLL.dll 都要放在可以看到的地方

UseDLL 的 Linker Option



# 使用不同的 name

# 檢視結果

注意: 原來的 Add 會被 forward 到 NewDLL中的