

1. Lời giới thiệu

Hiện nay, ngày càng nhiều mã độc được viết bằng C++ thay vì thuần C. Mà sự khác biệt giữa C++ và C khi lập trình chính là việc ứng dụng OOP. Điều này gây ra nhiều khó khăn khi reverse mã độc. Thông qua bài viết này tôi sẽ hướng dẫn các bạn reverse OOP trong C++ theo một cách súc tích và ngắn gọn nhất như mì ăn liền.

2. Phân tích

1. 2.1. Tổng quan

Để có thể đọc hiểu được bài viết này thì bắt buộc các bạn phải có kiến thức cơ bản về OOP được thể hiện bằng C++ nhé. Khi một mã nguồn viết theo phong cách OOP và được biên dịch sang mã máy thì các bạn có bao giờ thắc mắc liệu nó có khác gì so với các chương trình được viết bằng C thuần. Tôi sẽ giúp các bạn trả lời những câu hỏi sau khi phân tích chương trình được viết theo phong cách OOP:

1. Làm thế nào mà các hàm override ở lớp con được gọi trong chương trình hay nói rộng hơn là các lớp con khi được khởi tạo trên bộ nhớ sẽ chứa những thông tin gì bên trong.
2. Mối quan hệ giữa các lớp cha con được thể hiện ra sao trong chương trình
3. Cuối cùng sẽ là ứng dụng những thông tin mà chúng ta có được vào công cụ IDA khi phân tích.

Các bạn sử dụng chương trình được đính kèm và load vào IDA 32bit. Chương trình được build từ [link](#) sau. Nhưng các bạn khoan hãy xem source code nhé, vì khi reverse mã độc trên thực tế mình cũng đâu có source code đâu.

2. 2.2. Các trường bên trong lớp con khi được khởi tạo

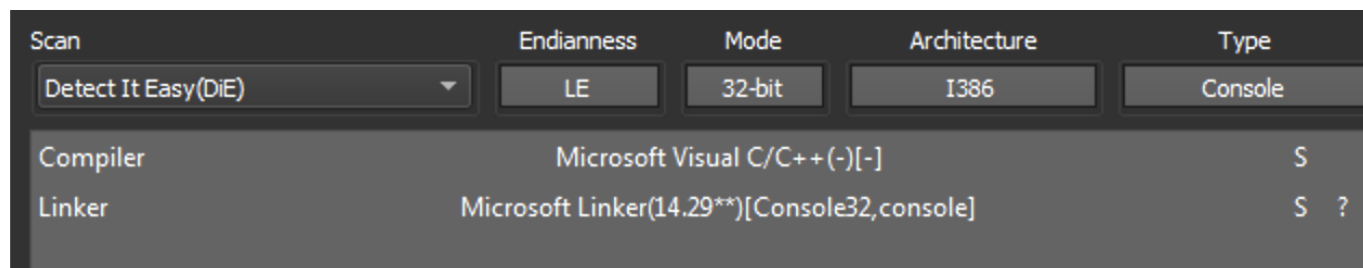
Khi load vào IDA, chúng ta sẽ có được kết quả như hình dưới. Chúng ta sẽ đi qua từng dòng mã lệnh Assembly để xem khi lớp con được khởi tạo trên bộ nhớ thì bao gồm những thông tin gì bên trong object đó.

```

.text:004010D0      = dword ptr -4
.text:004010D0 var_4      = dword ptr 8
.text:004010D0 argc      = dword ptr 0Ch
.text:004010D0 argv      = dword ptr 10h
.text:004010D0
.text:004010D0      push     ebp
.text:004010D1      mov      ebp, esp
.text:004010D3      push     ecx
.text:004010D4      push     esi
.text:004010D5      push     edi
.text:004010D6      push     14h          ; length of object is 14 bytes
.text:004010D8      call     ??2@YAPAXI@Z ; operator new(uint)
.text:004010DD      mov      edi, eax      ; edi hold address of object
.text:004010DF      add      esp, 4        ; clean stack
.text:004010E2      mov      ecx, edi      ; ecx hold address of object because of __thiscall
.text:004010E4      mov      [ebp+var_4], edi ; move address of object to local variable
.text:004010E7      mov      dword ptr [edi+8], offset ??_7Ex4@@@6B@ ; const Ex4::`vftable'
.text:004010EE      mov      dword ptr [edi], offset ??_7Ex5@@@6B@ ; const Ex5::`vftable'
.text:004010F4      mov      dword ptr [edi+8], offset ??_7Ex5@@@6B@_0 ; const Ex5::`vftable'
.text:004010FB      mov      dword ptr [edi+0Ch], 1 ; fourth field of object is 1
.text:00401102      mov      dword ptr [edi+10h], 2 ; fifth field of object is 2
.text:00401109      call     sub_401050     ; function overriding
.text:0040110E      mov      eax, [edi+8]   ; third field of object is address point to array of function overriding
.text:00401111      lea      ecx, [edi+8]   ; ecx hold address of object because of __thiscall
.text:00401114      call     dword ptr [eax] ; function overriding
.text:00401116      mov      eax, [edi]     ; first field of object is address point to array of function overriding
.text:00401118      mov      ecx, edi      ; ecx hold address of object because of __thiscall
.text:0040111A      call     dword ptr [eax+8] ; function overriding
.text:0040111D      push     14h          ; length of object
.text:0040111F      push     edi          ; address of object
.text:00401120      call     sub_40116E     ; delete object
.text:00401125      add      esp, 8        ; clean stack
.text:00401128      xor      eax, eax
.text:0040112A      pop      edi
.text:0040112B      pop      esi
.text:0040112C      mov      esp, ebp
.text:0040112E      pop      ebp
.text:0040112F      retn

```

Đầu tiên, ở địa chỉ 0x004010D8 là lệnh new một vùng nhớ có chiều dài 0x14 bytes. Sau đó, địa chỉ vùng nhớ này được đưa vào thanh ghi edi và ecx. Lí do mà chương trình làm vậy là vì thanh ghi edi được lựa chọn để sử dụng làm thanh ghi nền để truy xuất vào các trường khác trong object thông qua cộng 01 lượng offset. Trong khi đó, thanh ghi ecx sẽ nắm giữ địa chỉ object bởi vì chương trình mà chúng ta đang phân tích được build bằng MSVC của Visual Studio. Theo quy định của MSVC thì calling convention được sử dụng cho các hàm ảo sẽ là thiscall và thanh ghi ecx sẽ luôn luôn được sử dụng để nắm giữ địa chỉ của object.



Tiếp theo, chúng ta sẽ quan tâm ở địa chỉ 0x004010E7, 0x004010EE và 0x004010F4. Tại các địa chỉ này, chương trình sẽ đưa các vftable vào trong các trường của object theo thứ tự offset là 0x4, 0x0 và 0x4. Vftable các bạn cứ hiểu nôm na nó là cái bảng chứa địa chỉ các hàm ảo ở lớp cha và con. Thứ tự các hàm trong bảng được sắp xếp theo thứ tự xuất hiện các hàm ảo ở lớp cha rồi tới lớp con trong mã nguồn chương trình. Hay nói một cách khác là khi class B kế thừa class A, class B sẽ mở rộng từ class A. Vì thế các thuộc tính hàm ảo sẽ được thêm vào cùng theo thứ tự xuất hiện. Chúng ta sẽ xem qua ví dụ bên dưới

```

class A {
    int multiply(int x, int y) {
        return x * y;
    }

    virtual void printSomething () {
        printf("%s\n", "From class A with love");
    }
}

class B : A {
    virtual void printSomething () {
        printf("%s\n", "From class B with love");
    }

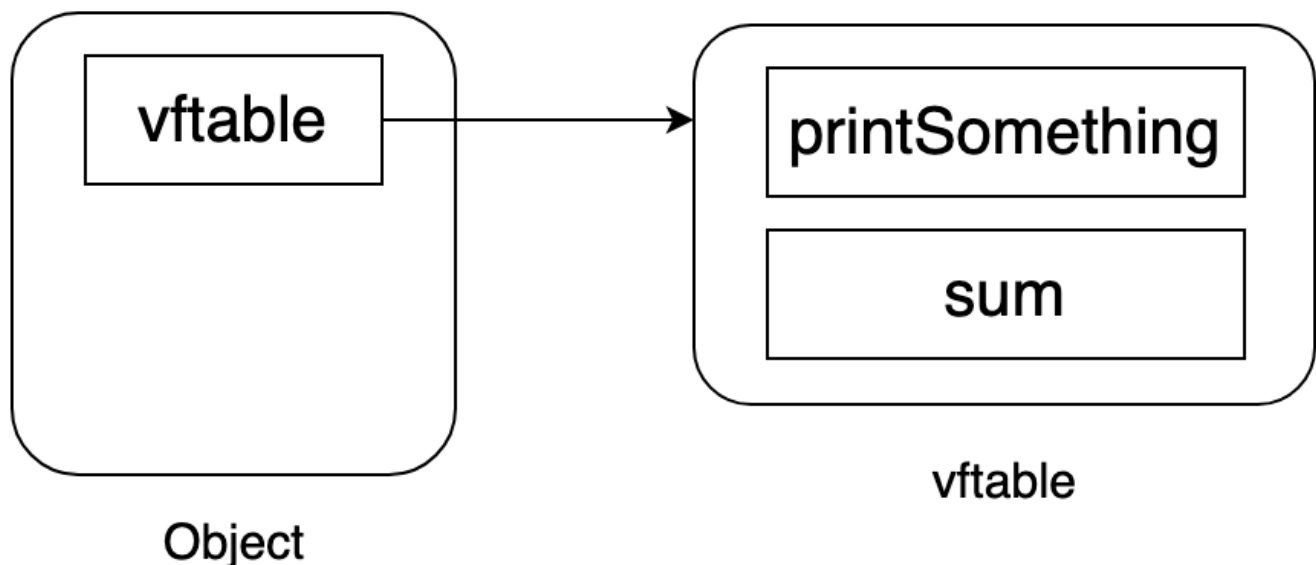
    virtual int sum (int x, int y) {
        return x + y;
    }
}

```

Class B thừa kế lớp A, trong đó hàm printSomething ở class B sẽ override hàm printSomething ở class A. Bởi vì hàm multiply ở class A không phải hàm ảo nên hàm này sẽ không xuất hiện trong bảng vtable. Hơn nữa, vì không có sự tồn tại hàm sum ở class A để hàm sum class B override như trường hợp hàm printSomething nên hàm này sẽ tồn tại độc lập trong bảng vtable. Bảng vtable của chúng ta sẽ như sau:

Offset	Function	Class
0x0	printSomething	B
0x4	sum	B

Để có thể gọi được các hàm ảo trong bảng vtable, chương trình cần xác định được vtable ở đâu trong object, sau đó cộng 01 lượng offset để gọi như 02 lệnh call ở địa chỉ 0x00401114 và 0x0040111A của chương trình thể hiện.



**call printSomething == call [vtable]
call sum == call [vtable + 0x4]**

Quay trở lại chương trình của chúng ta, dựa theo gợi ý của IDA, thì offset 0x0 của object là vtable của class Ex5 và 0x8 là vtable class Ex4. Tiếp theo đó ở địa chỉ 0x004010FB và 0x00401102, chương trình di chuyển giá trị 1 và 2 vào các offset 0xC và 0x10 của object. Sau cùng, object sẽ có layout như sau trên bộ nhớ:

Offset	Fields
0x0	vtable class Ex5
0x4	unknown
0x8	vtable class Ex4
0xc	1
0x10	2

Nếu chúng ta xem xét toàn bộ chương trình thì sẽ không thấy trường unknown tại offset 0x4 được sử dụng. Điều này đồng nghĩa với việc trường này chắc chắn không phải là vtable, nếu có thì chương trình bắt buộc phải khởi tạo giá trị cho nó. Bởi vì không có sự truy xuất đến trường này trong chương trình thì chúng ta sẽ coi đây chỉ là 01 trường bình thường như các trường tại offset 0xc và 0x10 của object. Dựa vào cách bố trí layout object khi được khởi tạo, các bạn nhìn thấy được quy luật sắp xếp của nó của nó rồi chứ. Giá trị của vtable được xếp đầu tiên, theo sau đó là các thuộc tính của class đó. Cuối cùng, các thuộc tính của class kế thừa sẽ được xếp cuối. Ta thấy có 02 vtable trong object vậy có thể suy ra có 02 class là Ex5 kế thừa Ex4. Tuy nhiên, trong trường hợp này có tổng cộng 03 class. Tôi biết được điều đó là nhờ vào RTTI.

3. 2.3. Cấu trúc RTTI trong chương trình

Quay lại câu hỏi thứ 02 đầu bài là mối quan hệ cha con giữa các lớp được thể hiện ra sao trong chương trình và làm sao tôi biết được có tổng cộng 03 class như cuối phần 2.2. Chúng tôi quay lại các vtable mà được đưa vào object tại các địa chỉ 0x004010E7, 0x004010EE và 0x004010F4 bằng lệnh mov của chương trình. Ta sẽ đi đến vtable đầu tiên tại vị trí 0x004021E0. Các bạn để ý một biến khác ở phía trên vtable ở vị trí 0x004021DC mà theo IDA có chú thích là const Ex4::`RTTI Complete Object Locator'.

```
.rdata:004021D8 ; const Ex5::`vtable'
.rdata:004021D8 ??_7Ex5@@6B@_0 dd offset sub_4010B0 ; DATA XREF: _main+24fo
.rdata:004021DC dd offset ??_R4Ex4@@6B@ ; const Ex4::`RTTI Complete Object Locator'
.rdata:004021E0 ; const Ex4::`vtable'
.rdata:004021E0 ??_7Ex4@@6B@ dd offset sub_401070 ; DATA XREF: _main+17fo
```

RTTICompleteObjectLocator là nơi chứa đựng các thông tin liên quan đến class đó cũng như dựa vào đó chúng ta sẽ biết được mối quan hệ với các class khác có trong chương trình. Cấu trúc của RTTICompleteObjectLocator như dưới. Trong đó, chúng ta sẽ quan tâm đến biến pTypeDescriptor và pClassDescriptor. Biến pTypeDescriptor sẽ chứa thông tin về class hiện tại còn pClassDescriptor chứa thông tin về các class mà nó kế thừa.

```
struct RTTICompleteObjectLocator {
    DWORD signature;
    DWORD offset;
    DWORD cdOffset;
    TypeDescriptor *pTypeDescriptor;
    RTTIClassHierarchyDescriptor *pClassDescriptor;
}
```

Biến pTypeDescriptor sẽ có kiểu dữ liệu là TypeDescriptor. Trong đó, trường name sẽ chứa tên của class đó. Ta truy xuất giá trị của biến name trên IDA và nhận thấy RTTICompleteObjectLocator tại địa chỉ chỉ 0x004021DC có tên class là Ex4.

```
struct TypeDescriptor
{
    const void* pVFTable;
    void* spare;
    char name[];
}
```

```
.data:004030A4 ; class Ex4 `RTTI Type Descriptor'
.data:004030A4 ??_R0?AVEx4@@@8 dd offset ??_7type_info@@6B@
.data:004030A4 ; DATA XREF: .rdata:004024C8fo
.data:004030A4 ; .rdata:Ex4::`RTTI Base Class Descriptor at (8,-1,0,64)'fo ...
.data:004030A4 ; reference to RTTI's vtable
.data:004030A8 dd 0 ; internal runtime reference
.data:004030AC aAvex4 db '.?AVEx4@@',0 ; type descriptor name
```

Đối với biến pClassDescriptor sẽ mang kiểu dữ liệu RTTIClassHierarchyDescriptor. Biến numBaseClasses sẽ chứa số lượng các hàm có trong vtable của class đó. Biến pBaseClassArray sẽ là con trỏ trỏ tới các phần tử RTTIBaseClassArray. Mà mỗi phần tử này sẽ đại diện chứa thông tin các class mà nó thừa kế. Trong đó, biến pTypeDescription sẽ trỏ đến TypeDescriptor của các class mà nó thừa kế. Dựa vào điều này ta sẽ có được tên class đó. Trong trường hợp hiện

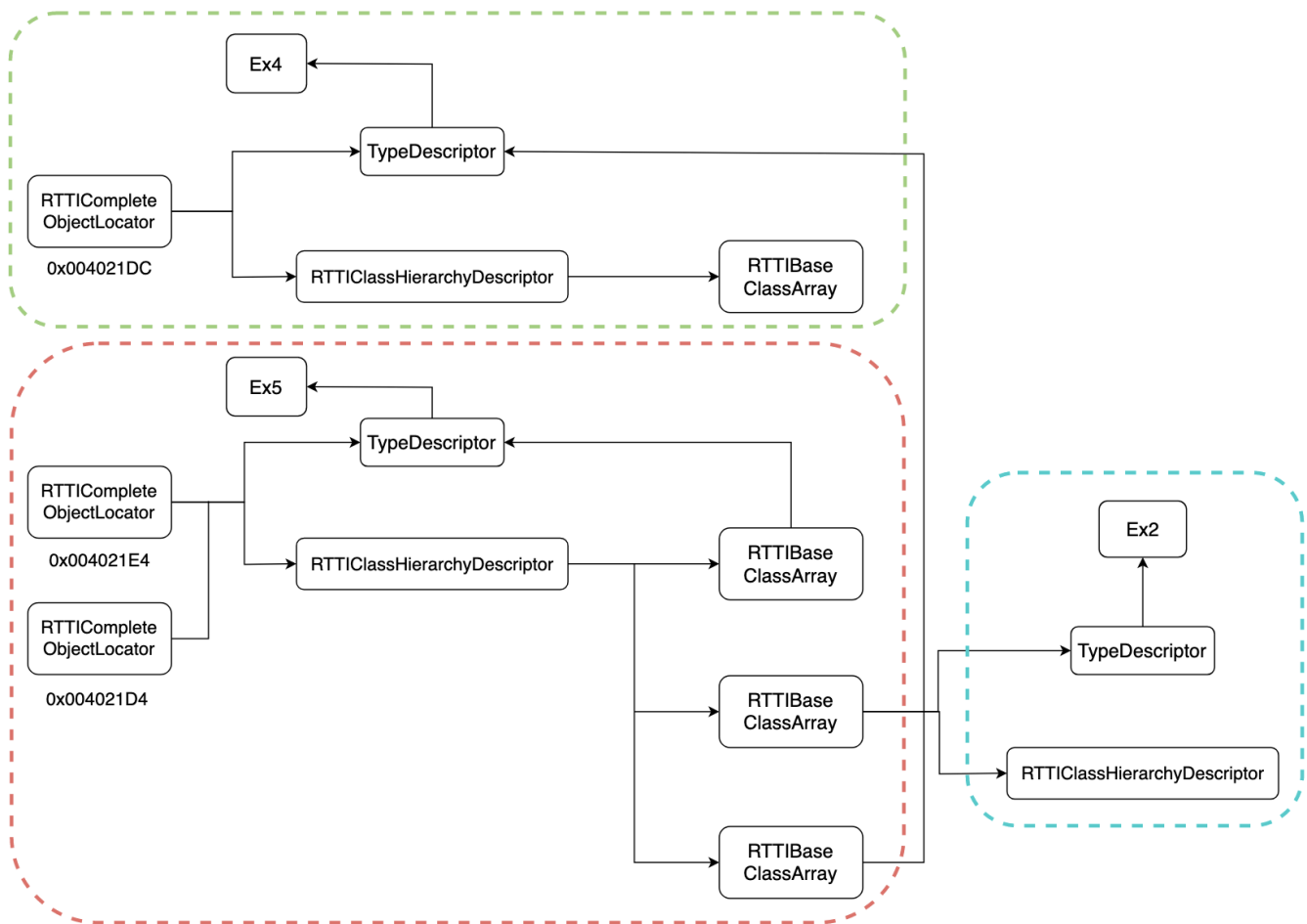
tại, class Ex4 chỉ có 01 hàm trong vtable và RTTIBaseClassArray trỏ tới chính nó. Như vậy, class Ex4 không kế thừa class nào.

```
struct RTTIClassHierarchyDescriptor {
    DWORD signature;
    DWORD attributes;
    DWORD numBaseClasses;
    DWORD pBaseClassArray;
}
```

```
.rdata:004024EC ; Ex4::`RTTI Class Hierarchy Descriptor'
.rdata:004024EC ??_R3Ex4@@@8      dd 0                ; DATA XREF: .rdata:004024CC↑o
.rdata:004024EC                ; .rdata:004024E8↑o ...
.rdata:004024EC                ; signature
.rdata:004024F0                dd 0                ; attributes
.rdata:004024F4                dd 1                ; # of items in the array of base classes
.rdata:004024F8                dd offset ??_R2Ex4@@@8 ; reference to the array of base classes
.rdata:004024FC ; Ex4::`RTTI Base Class Descriptor at (0, -1, 0, 64)'
```

```
struct RTTIBaseClassArray {
    DWORD pTypeDescription;
    DWORD numContainedBases;
    DWORD PMD.mdisp;
    DWORD PMD.pdisp;
    DWORD PMD.vdisp;
    DWORD attributes;
    DWORD pClassDescription;
}
```

Thế là xong một class, các bạn thực hiện điều tương tự đối với 02 RTTICompleteObjectLocator còn lại ở vị trí 0x004021E4 và 0x004021D4 . Nếu các bạn parse thông tin đúng, chúng ta sẽ thu được sơ đồ về mối quan giữa các class như hình dưới. Như vậy, class Ex5 kế thừa 02 class là Ex4 và Ex2.



Trong đó, vtable của Ex4 đã được chương trình đưa vào object tại địa chỉ 0x004010E7, nhưng vì ở class Ex5 có kế thừa lại hàm này nên sẽ được ghi đè vtable thêm lần nữa. Đó là lí do tại sao bạn lại thấy chương trình đưa vtable 02 lần cùng một vị trí trong object. Tuy nhiên, vtable của class Ex2 chúng ta lại không thấy chương trình đựng tới, thật ra nó được gộp chung với vtable của class Ex5.

4. 2.3. Reverse OOP in IDA

Đây có lẽ mà các bạn mong đợi nhất. Tổng hợp thông tin lại mà chúng ta có được cho tới hiện tại. Class Ex5 kế thừa 02 class theo thứ tự Ex2 và Ex4. Trong đó, cuối object có thêm 02 thuộc tính khác tại offset 0xc và 0x10, 02 thuộc tính có thể thuộc class Ex4 hoặc Ex5. Ở đây, chúng ta sẽ giả dụ 02 thuộc tính này thuộc về class Ex5. Trong bảng vtable của class Ex2 + Ex5 có 03 hàm ảo, còn class Ex4 chỉ có 01 hàm ảo. Như vậy, ta sẽ cần tạo ra các struct như sau trong IDA.

```
struct Ex4 {
    Ex4_vtbl *__vtable;
}
```

```
struct Ex2 {  
    Ex5_vtbl *__vftable;  
    DWORD var1;  
}
```

```
struct Ex5 : Ex2, Ex4 {  
    DWORD var2;  
    DWORD var3;  
}
```

```
struct Ex5_vtbl {  
    int (__thiscall *sum)(Ex5 *this);  
    void (__thiscall *Print1)(Ex5 *this);  
    void (__thiscall *Print2)(Ex5 *this);  
}
```

```
struct Ex4_vtbl {  
    void (__thiscall *Print3)(Ex4 *this);  
}
```

Đối với struct chứa các hàm ảo của mỗi class, các bạn cần ghi đúng cú pháp là `nameOfClass_vtbl`. Trong khi đó, phần khai báo con trỏ vtable để trỏ tới bảng đó bắt buộc phải có tên là `nameOfClass_vtbl *__vftable`. Vì các hàm trong bảng vtable là các hàm ảo nên sẽ có calling convention là `thiscall` theo như quy định của MSVC. Tham số đầu tiên ở mỗi hàm cũng chính là địa chỉ trỏ tới object trên bộ nhớ, cũng chính là thanh ghi `ecx` nắm giữ như tôi đã trình bày ở phía trên. Đối với khai báo struct `Ex5` như trên, chúng ta sẽ tách ra thành 02 struct nhỏ rồi dùng phép kế thừa đối với 02 struct đó. Bởi vì, nếu khai báo chung vô struct `Ex5` sẽ có 02 biến vtable, như vậy là không hợp lệ. Các bạn mở sang tab Local Type trong IDA rồi gõ các struct trên vào và tiến hành ép kiểu cho biến `v4` như hình dưới. Nếu thành công, các bạn sẽ có kết quả như dưới đây.


```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    _DWORD *v4; // [esp+8h] [ebp-4h]

    v4 = operator new(0x14u);
    v4[2] = &Ex4::`vftable';
    *v4 = &Ex5::`vftable';
    v4[2] = &Ex5::`vftable';
    v4[3] = 1;
    v4[4] = 2;
    sub_401050(v4);
    (*v4[2])(v4 + 2);
    (*(v4 + 8))(v4);
    sub_40116E(v4);
    return 0;
}

```

```

int __cdecl main(int argc, const char **argv, const char **envp)
{
    Ex5 *v4; // [esp+8h] [ebp-4h]

    v4 = operator new(0x14u);
    v4->Ex4::__vftable = &Ex4::`vftable';
    v4->Ex2::__vftable = &Ex5::`vftable';
    v4->Ex4::__vftable = &Ex5::`vftable';
    v4->var2 = 1;
    v4->var3 = 2;
    sub_401050(v4);
    v4->Print3(&v4->Ex4);
    v4->Print2(v4);
    sub_40116E(v4);
    return 0;
}

```

Đối với hàm sub_401050, các bạn quan sát lại vftable ở địa chỉ 0x004021E8, rõ ràng hàm này nằm trong bảng đó nhưng chương trình lại không dùng vftable để gọi nó. Mà để thể hiện cách gọi OOP như mã nguồn gốc, chúng ta sẽ sử dụng mangled name để ép nó về. Thay vì phải ngồi đọc document để hiểu cú pháp mangled name, chúng ta sẽ sử dụng đến plugin Classy. Các bạn bấm vào nút Add rồi gõ tên Ex5 vào. Sau đó, các bạn kéo tab Classy song song với tab Disassembly như hình dưới.

```

.rdata:004021E0 ; const Ex4: `vftable'
.rdata:004021E0 ??_7Ex4@@6B@ dd offset sub_401070 ; DATA XREF: _main+17fo
.rdata:004021E4 dd offset ??_R4Ex5@@6B@ ; const Ex5: `RTTI Compl
.rdata:004021E8 ; const Ex5: `vftable'
.rdata:004021E8 ??_7Ex5@@6B@ dd offset sub_401040 ; DATA XREF: _main+1Efo
.rdata:004021EC dd offset sub_401050
.rdata:004021F0 dd offset sub_401090
.rdata:004021F4 ; Debug Directory entries
.rdata:004021F4 dd 0 ; Characteristics
.rdata:004021F8 dd 6247D4CBh ; TimeDateStamp: Sat Apr
.rdata:004021FC dw 0 ; MajorVersion
.rdata:004021FE dw 0 ; MinorVersion
.rdata:00402200 dd 2 ; Type: IMAGE_DEBUG_TYPE
.rdata:00402204 dd 46h ; SizeOfData
.rdata:00402208 dd rva asc_402584 ; AddressOfRawData
.rdata:0040220C dd 1984h ; PointerToRawData
.rdata:00402210 dd 0 ; Characteristics
.rdata:00402214 dd 6247D4CBh ; TimeDateStamp: Sat Apr
.rdata:00402218 dw 0 ; MajorVersion
.rdata:0040221A dw 0 ; MinorVersion
.rdata:0040221C dd 0Ch ; Type: IMAGE_DEBUG_TYPE
.rdata:00402220 dd 14h ; SizeOfData
.rdata:00402224 dd rva unk_4025CC ; AddressOfRawData
.rdata:00402228 dd 19CCCh ; PointerToRawData
.rdata:0040222C dd 0 ; Characteristics
.rdata:00402230 dd 6247D4CBh ; TimeDateStamp: Sat Apr
.rdata:00402234 dw 0 ; MajorVersion
.rdata:00402236 dw 0 ; MinorVersion
.rdata:00402238 dd 0Dh ; Type: IMAGE_DEBUG_TYPE
.rdata:0040223C dd 2A4h ; SizeOfData
.rdata:00402240 dd rva a6ctl ; AddressOfRawData
.rdata:00402244 dd 19E0h ; PointerToRawData

```

Ex5

Name: Ex5
Base class: None
Derived classes: None
Struct: Not set
VTable: Not set

ID	Address	Function	Type

Address	Function

Add Remove

Sau đó, các bạn di chuyển đến vftable của Ex5 ở vị trí 0x004021E8, bôi chuột từ vị trí 0x004021E8 đến 0x004021F4 và bấm nút Set kế bên "VTable: Not set" để thêm các hàm này vào class Ex5.

```

.rdata:004021E0 ; const Ex4: `vftable'
.rdata:004021E0 ??_7Ex4@@6B@ dd offset sub_401070 ; DATA XREF: _main+17fo
.rdata:004021E4 dd offset ??_R4Ex5@@6B@ ; const Ex5: `RTTI Compl
.rdata:004021E8 ; const Ex5: `vftable'
.rdata:004021E8 ??_7Ex5@@6B@ dd offset _ZN3Ex53vf0Ev ; DATA XREF: _main+1Efo
.rdata:004021EC dd offset _ZN3Ex53vf4Ev
.rdata:004021F0 dd offset _ZN3Ex53vf8Ev
.rdata:004021F4 ; Debug Directory entries
.rdata:004021F4 dd 0 ; Characteristics
.rdata:004021F8 dd 6247D4CBh ; TimeDateStamp: Sat Apr
.rdata:004021FC dw 0 ; MajorVersion
.rdata:004021FE dw 0 ; MinorVersion
.rdata:00402200 dd 2 ; Type: IMAGE_DEBUG_TYPE
.rdata:00402204 dd 46h ; SizeOfData
.rdata:00402208 dd rva asc_402584 ; AddressOfRawData
.rdata:0040220C dd 1984h ; PointerToRawData
.rdata:00402210 dd 0 ; Characteristics
.rdata:00402214 dd 6247D4CBh ; TimeDateStamp: Sat Apr
.rdata:00402218 dw 0 ; MajorVersion
.rdata:0040221A dw 0 ; MinorVersion
.rdata:0040221C dd 0Ch ; Type: IMAGE_DEBUG_TYPE
.rdata:00402220 dd 14h ; SizeOfData
.rdata:00402224 dd rva unk_4025CC ; AddressOfRawData
.rdata:00402228 dd 19CCCh ; PointerToRawData
.rdata:0040222C dd 0 ; Characteristics
.rdata:00402230 dd 6247D4CBh ; TimeDateStamp: Sat Apr
.rdata:00402234 dw 0 ; MajorVersion
.rdata:00402236 dw 0 ; MinorVersion
.rdata:00402238 dd 0Dh ; Type: IMAGE_DEBUG_TYPE
.rdata:0040223C dd 2A4h ; SizeOfData
.rdata:00402240 dd rva a6ctl ; AddressOfRawData
.rdata:00402244 dd 19E0h ; PointerToRawData

```

Ex5

Name: Ex5
Base class: None
Derived classes: None
Struct: Not set
VTable: 0x4021E8 - 0x4021F4

ID	Address	Function	Type
0	0x401040	void Ex5::vf0()	virtual
1	0x401050	void Ex5::vf4()	virtual
2	0x401090	void Ex5::vf8()	virtual

Address	Function

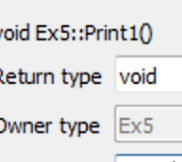
Add Remove

Cuối cùng, các bạn đổi tên hàm lại là Print1 như chúng ta khai báo ở struct Ex5_vtbl bằng cách double click vào row thứ 2 (ID: 1) và sửa lại.

VTable: 0x4021E8 - 0x4021F4

ID	Address	
0	0x401040	void Ex5::vf0()
1	0x401050	void Ex5::vf4()
2	0x401090	void Ex5::vf8()

Address



Set function signature

void Ex5::Print1()

Return type: void

Owner type: Ex5

Name: Print1

Arguments:

Const: ☐

Ctor: C1: complete

Dtor: D1: complete

Status: Valid

Mangled: _ZN3Ex56Print1Ev

OK Cancel

Kết quả cuối cùng mà chúng ta có được như sau.

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
    Ex5 *v4; // [esp+8h] [ebp-4h]

    v4 = operator new(0x14u);
    v4->Ex4::__vftable = &Ex4::__vftable';
    v4->Ex2::__vftable = &Ex5::__vftable';
    v4->Ex4::__vftable = &Ex5::__vftable';
    v4->var2 = 1;
    v4->var3 = 2;
    Ex5::Print1();
    v4->Print3(&v4->Ex4);
    v4->Print2(v4);
    sub_40116E(v4);
    return 0;
}
```