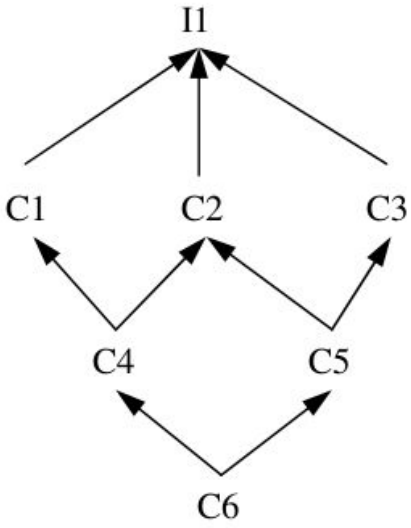
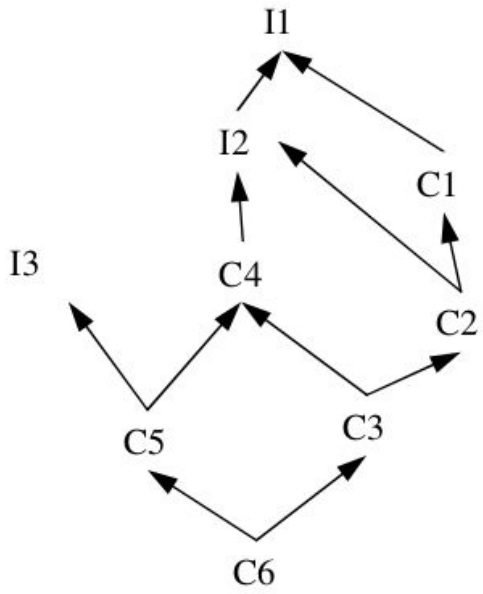
Άσκηση 3

Μιχαήλ Μυλωνάκης

16 Δεκεμβρίου 2015

1. *Δείξτε τη μορφή αντικειμένων στη μνήμη (σύμφωνα με τους συνηθισμένους κανόνες για πολλαπλή κληρονομικτότητα χωρίς βελτιστοποιήσεις) για τη χαμηλότερη υπο-κλάση στα παρακάτω διαγράμματα. Σε περίπτωση πολλαπλής κληρονομικότητας χρησιμοποιήστε την αριστερή υπερ-κλάση σαν πρωταρχική.*





Στο πρώτο σχήμα, παρατηρούμε ότι υπάρχουν 4 μονομάτια προς την κορυφή της ιεραρχίας μας. Περιμένουμε λοιπόν το αντικείμενο C6 να έχει 4 δείκτες σε Vtables:

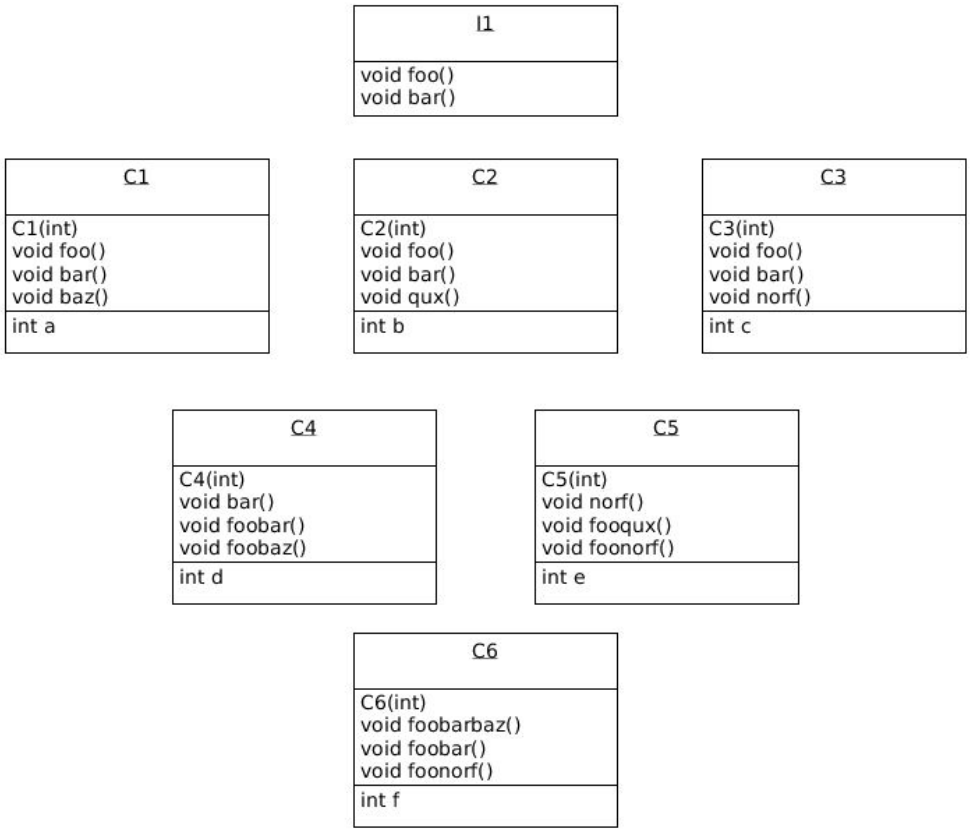
I1 / C1 / C4 / C6 -> vtable
<<C1 fields>>
I1 / C2 / C4 -> vtable
<<C2 fields>>
<<C4 fields>>
I1 / C2 / C5 -> vtable
<<C2 fields>>
I1 / C3 / C5 -> vtable
<<C3 fields>>
<<C5 fields>>
<<C6 fields>>

Για την δεύτερη ιεραρχία κλάσεων, παρατηρούμε ότι έχουμε 5 μονοπάτια προς τις κορυφές της ιεραρχίας, άρα περιμένουμε 5 δείκτες σε Vtables:

I3 / C5 / C6 -> vtable
I1 / I2 / C4 / C5 -> vtable
<<C4 fields>>
<<C5 fields>>
I1 / I2 / C4 / C3 -> vtable
<<C4 fields>>
I1 / I2 / C2 / C3 -> vtable
I1 / C1 / C2 / C3 -> vtable
<<C1 fields>>
<<C2 fields>>
<<C3 fields>>
<<C6 fields>>

2. *Γράψτε ένα πρόγραμμα C++ για να ανακαλύψετε τη μορφή αντικειμένων στη μνήμη κάτω από πολλαπλή κληρονομικότητα. Είναι η μορφή που βλέπετε ίδια με αυτή που είδαμε στο μάθημα; Με βάση το πλήθος των δεικτών σε v-tables κλαντε δοκιμές για να διαπιστώσετε αν ο compiler προσπαθεί να σμίξει v-tables (δηλαδή αν προσπαθεί να δώσει offsets σε μεθόδους που να μην συμπίπτουν για διαφορετικές υπερ-κλάσεις). Επίσης προσπαθείστε να ανακαλύψετε πως ο compiler αλλάζει τον δείκτη "this" για μεθόδους που κληρονομούνται ή υποσκελίζονται. (Π.χ. Βλέπετε ότι η ίδια μέθοδος εμφανίζεται με λίγο διαφορετική διεύθυνση στον πρωταρχικό και το μη-προταχρικό v-table; Βλέπετε οι v-tables να περιέχουν και Offsets για ρύθμιση του δείκτη "this";)*

Τα αποτελέσματα που θα παρουσιαστούν βασίζονται στην ιεραρχία κλάσεων του 1ου σχήματος του προηγούμενου ερωτήματος. Πιο συγκεκριμένα προστέθηκαν οι ακόλουθες μέδοδοι / class fields στις κλάσεις:



Εικόνα 1. Class Diagram

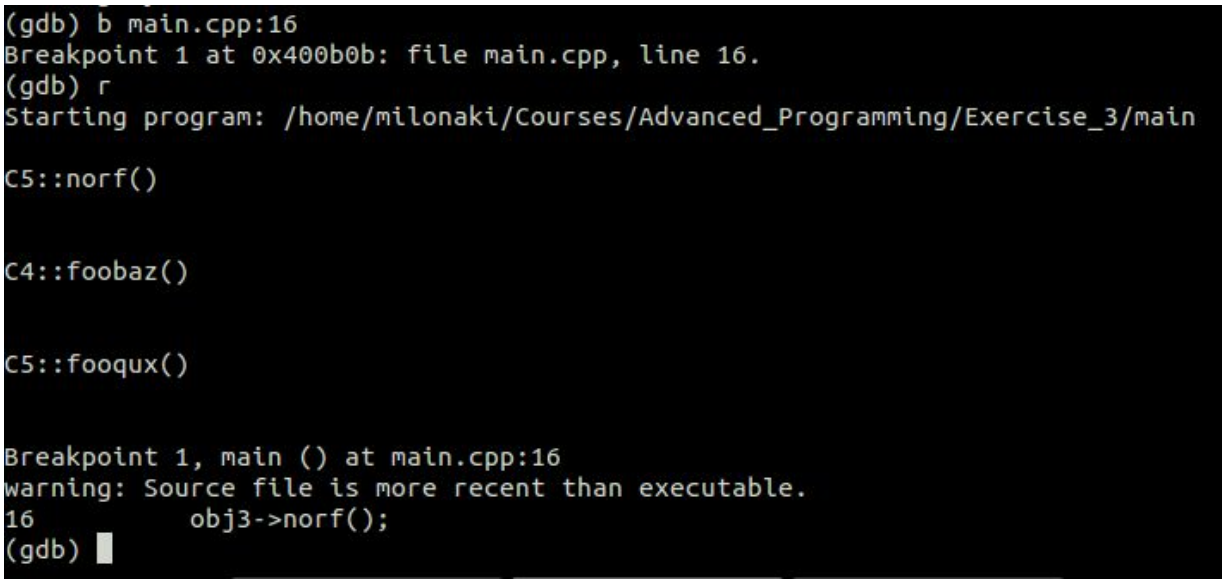
Η main δημιουργεί αντικείμενα από όλες τις κλάσεις, και καλεί ενδεικτικά μερικές μεθόδους:

main.cpp

```cpp
#include "C6.hpp"
#include "C5.hpp"
#include <iostream>


int main() {
    C6 obj6 = C6(15);
    C5 obj5 = C5(9);
    C4 obj4 = C4(3);
    C1 obj1 = C4(8);

    obj6.norf();
    obj6.foobaz();
    obj6.fooqux();
    C3 *obj3 = &obj6;
    obj3->norf();
    obj6.baz();

    return 0;
}
```

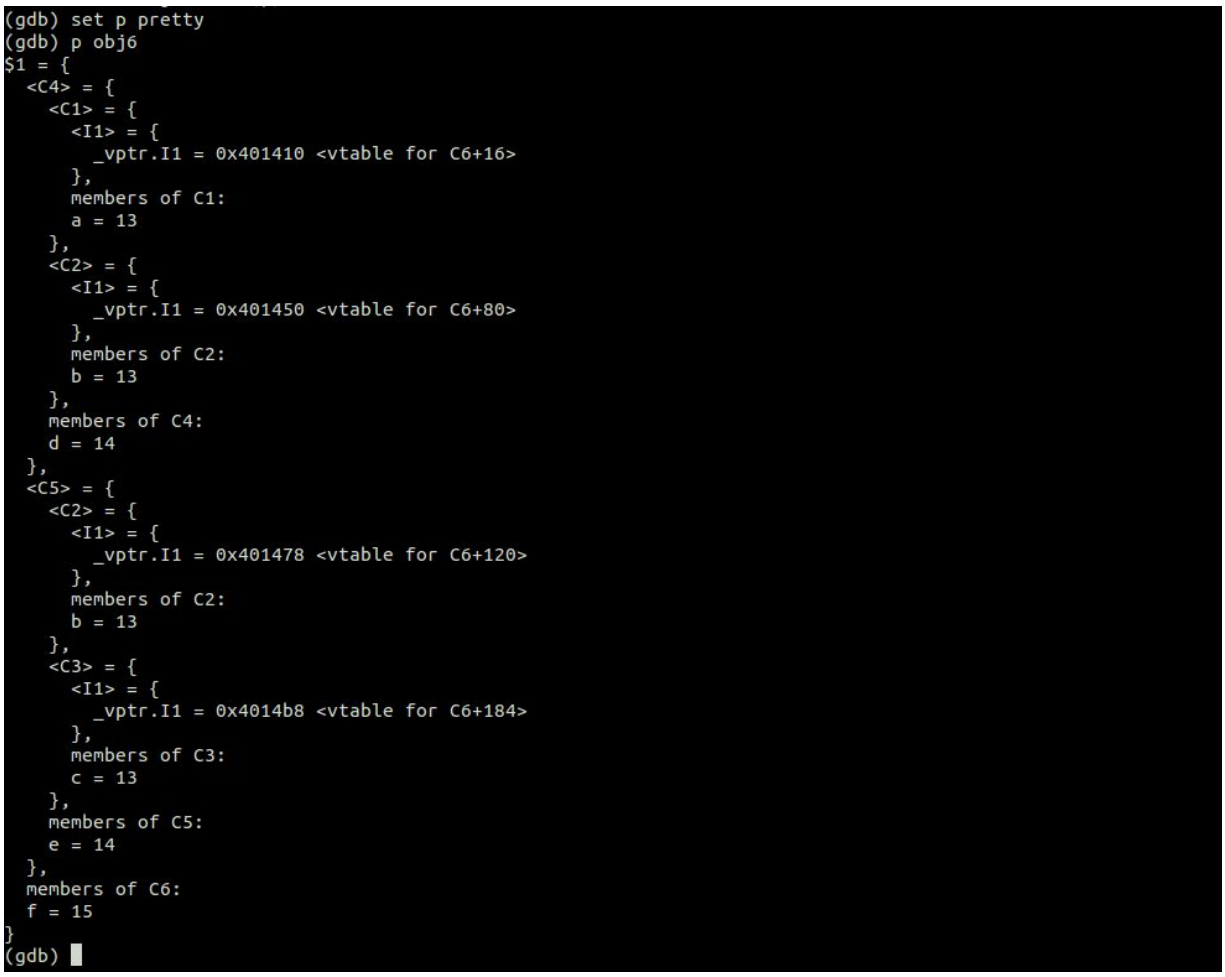
Εικόνα 2. main.cpp

Τα συμπεράσματα που παρουσιάζονται, εξήσθησαν μέσα από την ανάλυση με debugger (gdb). Αρχικά βάλαμε ένα breakpoint στην γραμμή 17 του main.cpp , έτσι ώστε να είναι όλα τα αντικείμενά μας ζωντανά (In scope).



Εικόνα 3. gdb breakpiont

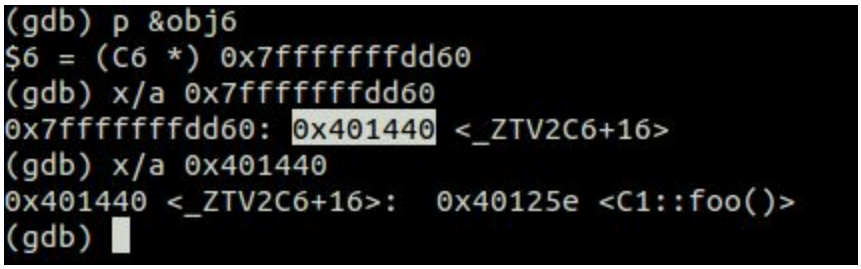
Έπειτα, βλέπουμε το layout του αντικειμένου obj6 στην μνήμη:



Εικόνα 4. Obj6 layout

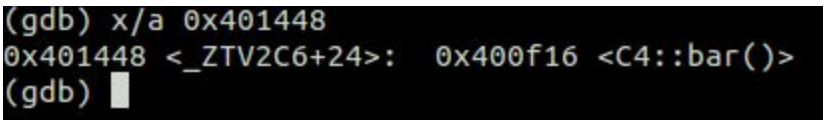
Το Layout είναι ακριβώς όπως το περιμέναμε (και παρουσιάστηκε στο 1ο υποερώτημα). Βλέπουμε ότι υπάρχουν 4 δείκτες σε vtable, και ότι ένα αντικείμενο C6 μπορεί να παίξει τον ρόλο του C1 με έναν τρόπο, του C2 με δύο τρόπους, του I1 με 4 τρόπους κλπ. Σύμφωνα με τα παραπάνω , το πρώτο πεδίο στο αντικείμενο obj6, είναι pointer σε vtable. Για να επαληθεύσουμε τον συλλογισμό αυτόν, εκτυπώσαμε το περιεχόμενο της διεύθυνσης του obj6, αναμένουμε να είναι ο pointer στο vtable, και μετά εκτυπώνουμε το περιεχόμενο του pointer, και αναμένουμε να

είναι ένας δείκτης στην πρώτη συνάρτηση του vtable, λογικά την C1::foo, αφού ούτε ή C4 ούτε η C6 την κάνουν override:



Εικόνα 5. C1::foo()

Δουλεύουμε σε 64-bit αρχιτεκτονική, οπότε στην επόμενη θέση μνήμης του vtable, την 0x401418 περιμένουμε να δούμε την C4::bar, καθώς υποσκελίζει την C1::bar :



Εικόνα 6. C4::bar()

Ομοίως εξετάζουμε τις επόμενες θέσεις του πίνακα vtable. Συνολικά μία προς μία οι θέσεις του vtable φαίνονται παρακάτω:

```
(gdb) x/a 0x401440
0x401440 <_ZTV2C6+16>:  0x40125e <C1::foo()>
(gdb) x/a 0x401448
0x401448 <_ZTV2C6+24>:  0x400f16 <C4::bar()>
(gdb) x/a 0x401450
0x401450 <_ZTV2C6+32>:  0x4012b6 <C1::baz()>
(gdb) x/a 0x401458
0x401458 <_ZTV2C6+40>:  0x400c8a <C6::foobar()>
(gdb) x/a 0x401460
0x401460 <_ZTV2C6+48>:  0x400f74 <C4::foobaz()>
(gdb) x/a 0x401468
0x401468 <_ZTV2C6+56>:  0x400cb6 <C6::foonorf()>
```

Εικόνα 7. obj6 vtable

Παρατηρούμε ότι οι μέθοδοι βρίσκονται σε συνεχόμενες θέσεις του πίνακα vtable του obj6 - δεν έχει εισαχθεί δηλαδή κάποιο offset, με σκοπό δηλαδή να σμίξει κάποιο άλλο vtable μαζί με αυτό.

Σύμφωνα με τα όσα έχουμε υποθέσει, στην επόμενη θέση μνήμης του obj6, περιμένουμε να βρούμε τα παιδία της κλάσης C1. Η κλάση C1 έχει αρχικοποιηθεί από τον constructor της C4, που με την σειρά του έχει κληθεί από τον constructor της C6. Αρχικοποιήσαμε την C6 με την τιμή 15. Η κάθε κλάση στην ιεραρχία, καλεί τον constructor της super - κλάσης βάζοντας σαν όρισμα το όρισμα του δικού της constructor - 1. Στην περίπτωσή μας οι C6, C4:

```
1 #include "C4.hpp"
2 #include <iostream>
3
4
5 C4::C4(long int d) : C1(d-1), C2(d-1)
6 {
7     this->d = d;
8 }
```

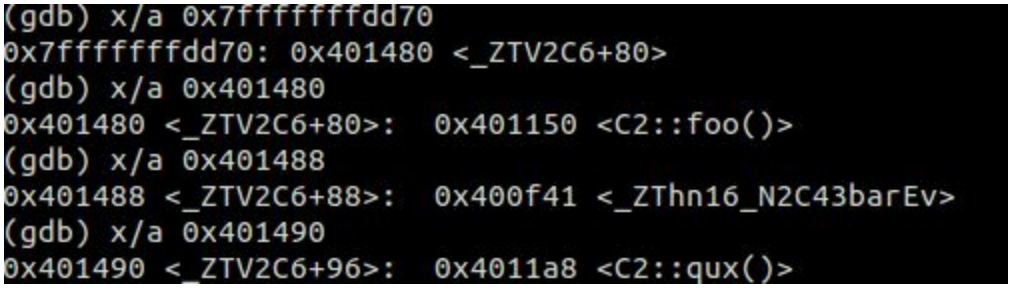
```
1 #include "C6.hpp"
2 #include <iostream>
3
4
5 C6::C6(long int f) : C4(f-1), C5(f-1)
6 {
7     this->f = f;
8 }
```

Εικόνα 8. C6, C4 Constructors

Άρα στα πεδία της C1 περιμένουμε να δούμε την τιμή 15 - 2, δηλαδή 13. Επαληθεύουμε μέσω του gdb:

Εικόνα 9. C1 fields 0xd = 13 decimal

Στην επόμενη θέση μνήμης του obj6, δηλαδή την 0x7fffffffdd70, περιμένουμε να δούμε έναν άλλον δείκτη σε vtable, που αφορά τα I1 / C2 / C4 (η αρχική διεύθυνση του obj6 ήταν η 0x7fffffffdd60, αυτός ήταν και ο πρώτος δείκτης στον vtable που αντιστοιχεί στα I1 / C1 / C4 / C6):



Εικόνα 10. I1 / C2 /C4 vtable

Η πρώτη θέση του vtable, δείχνει στην C2::foo() όπως αναμέναμε. Η επόμενη θέση φαίνεται να δείχνει σε μια συνάρτηση με περίεργο όνομα. Σύμφωνα με αυτά που έχουμε πει στην θεωρία, για να δουλέψει η C4::bar() η οποία έχει υποσκελιστεί σε αυτό το v-table (που παίζει το ρόλο ενός αντικειμένου τύπου C2) χρειάζεται ρύθμιση του this. Υποπτευόμαστε λοιπόν ότι η συνάρτηση αυτή παίζει τον ρόλο της stub συνάρτησης (στην βιβλιογραφία συναντάται συχνά ως

“thunk”), έτσι ώστε το this Να ρυθμιστεί ώστε να δείχνει στο obj6. Κάνουμε disassemble το σώμα την συνάρτησης μέσω του gdb και βλέπουμε:

```
(gdb) disassemble 0x400f41
Dump of assembler code for function _ZThn16_N2C43barEv:
   0x0000000000400f41 <+0>:     sub    $0x10,%rdi
   0x0000000000400f45 <+4>:     jmp    0x400f16 <C4::bar()>
End of assembler dump.
```

Εικόνα 11. Thunk Disassemble - offset adjusment demonstration

Η υπόθεσή μας επαληθεύεται: Ο rdi περιέχει το this, και αφαιρείτε το 0x10 από αυτόν, οπότε πρακτικά δείχνει 2 θέσεις μνήμης πίσω από το τρέχων vtable (δηλαδή το τρέχων this). Να θυμίσουμε ότι στην προηγούμενη θέση μνήμης υπήρχε το class field (long int a) της C1, και στην προηγούμενη από αυτήν θέση (την 0x7fffffffdd60) είναι ο vtable για το obj6. Συνεπώς η συνάρτηση αυτή ρυθμίζει το this ώστε να φαίνεται ως obj6, και έπειτα καλείται η C4::bar().

Μετά από τον I1 / C2 / C4 vtable, στην διεύθυνση 0x7fffffffdd78, 0x7fffffffdd80 περιμένουμε να δούμε τα C2 fileds, C4 fields. Όπως εξηγήσαμε παραπάνω, στα C2 fields περιμένουμε να δούμε έναν int με τιμή 13, και στα C4 fields περιμένουμε να δούμε έναν int με την τιμή 14. Επαληθεύουμε μέσω του gdb:

```
(gdb) x/a 0x7fffffffdd78
0x7fffffffdd78: 0xd
(gdb) x/a 0x7fffffffdd80
0x7fffffffdd80: 0xe
(gdb)
```

Εικόνα 12. C2 / C4 class fields

Στην επόμενη θέση μνήμης, δηλαδή την 0x7fffffffdd88, περιμένουμε να δούμε τον I1 / C2 / C5 vtable. Ακολουθώντας την προηγούμενη μεθοδολογία εξετάζουμε τον συγκεκριμένο vtable:

```
(gdb) x/a 0x4014a8
0x4014a8 <_ZTV2C6+120>: 0x401150 <C2::foo()>
(gdb) x/a 0x4014b0
0x4014b0 <_ZTV2C6+128>: 0x40117c <C2::bar()>
(gdb) x/a 0x4014b8
0x4014b8 <_ZTV2C6+136>: 0x4011a8 <C2::qux()>
(gdb) x/a 0x4014c0
0x4014c0 <_ZTV2C6+144>: 0x400dd0 <C5::norf()>
(gdb) x/a 0x4014c8
0x4014c8 <_ZTV2C6+152>: 0x400e02 <C5::fooqux()>
(gdb) x/a 0x4014d0
0x4014d0 <_ZTV2C6+160>: 0x400ce1 <_ZThn40_N2C67foonorfEv>
```

Εικόνα 13. I1 / C2 / C5 vtable.

Ενδιαφέρον παρουσιάζει η τελευταία θέση του vtable. Φαίνεται να είναι thunk συνάρτηση για την ρύθμιση του this, έτσι ώστε να κληθεί η overriden συνάρτηση C5::foonorf(). Αναμένουμε να αφαιρεθούν όσες θέσεις μνήμης απέχει ο τρέχων vtable (το τρέχων this) από το obj6, δηλαδή 5 θέσεις μνήμης. Κάνουμε disassemble την thunk συνάρτηση:

```
(gdb) disassemble 0x400ce1
Dump of assembler code for function _ZThn40_N2C67foonorfEv:
   0x0000000000400ce1 <+0>:     sub    $0x28,%rdi
   0x0000000000400ce5 <+4>:     jmp    0x400cb6 <C6::foonorf()>
End of assembler dump.
(gdb)
```

Εικόνα 14. Thunk function disassemble.

Η υπόθεση επαληθεύεται, καθώς αφαιρείται το 0x28, άρα 40 decimal == 5 bytes.

Ομοίως περιμένουμε τα C2 fields Μετά (στην διεύθυνση 0x7fffffffdd90), δηλαδή 13 έπειτα ο vtable για τα I1 / C3 / C5 μετά τα C3 class fields (== 13), έπειτα τα C5 fields (== 14) και τελικά τα C6 fields (== 15). Συνολικά βλέπουμε από τον gdb:

```
(gdb) x/a 0x7fffffffdd90
0x7fffffffdd90: 0xd
(gdb)
```

Εικόνα 15.

Ακολουθεί ο τελευταίος vtable, που αντιστοιχεί στα I1 / C3 / C5

```
(gdb) x/a 0x7fffffffdd98
0x7fffffffdd98: 0x4014e8 <_ZTV2C6+184>
(gdb) x/a 0x4014e8
0x4014e8 <_ZTV2C6+184>: 0x40102a <C3::foo()>
(gdb) x/a 0x4014f0
0x4014f0 <_ZTV2C6+192>: 0x401056 <C3::bar()>
(gdb) x/a 0x4014f8
0x4014f8 <_ZTV2C6+200>: 0x400dfb <_ZThn16_N2C54norfEv>
(gdb)
```

Εικόνα 16. I1 / C3 / C5 vtable

Βλέπουμε και εδώ την συνάρτηση ρύθμισης του this για την Overriden C5::norf().

Ακολουθούν τα παιδία της C3, C5, C6

```
(gdb) x/a 0x7fffffffdda0
0x7fffffffdda0: 0xd
(gdb) x/a 0x7fffffffdda8
0x7fffffffdda8: 0xe
(gdb) x/a 0x7fffffffddb0
0x7fffffffddb0: 0xf
(gdb)
```

Εικόνα 17. C3 / C5 / C6 class fields

Συνοψίζοντας , βλέπουμε όλα τα vtables του obj6:

```
(gdb) info vtbl obj6
vtable for 'C6' @ 0x401440 (subobject @ 0x7fffffffdd60):
[0]: 0x40125e <C1::foo()>
[1]: 0x400f16 <C4::bar()>
[2]: 0x4012b6 <C1::baz()>
[3]: 0x400c8a <C6::foobar()>
[4]: 0x400f74 <C4::foobaz()>
[5]: 0x400cb6 <C6::foonorf()>

vtable for 'C2' @ 0x401480 (subobject @ 0x7fffffffdd70):
[0]: 0x401150 <C2::foo()>
[1]: 0x400f41 <non-virtual thunk to C4::bar()>
[2]: 0x4011a8 <C2::qux()>

vtable for 'C5' @ 0x4014a8 (subobject @ 0x7fffffffdd88):
[0]: 0x401150 <C2::foo()>
[1]: 0x40117c <C2::bar()>
[2]: 0x4011a8 <C2::qux()>
[3]: 0x400dd0 <C5::norf()>
[4]: 0x400e02 <C5::fooqux()>
[5]: 0x400ce1 <non-virtual thunk to C6::foonorf()>

vtable for 'C3' @ 0x4014e8 (subobject @ 0x7fffffffdd98):
[0]: 0x40102a <C3::foo()>
[1]: 0x401056 <C3::bar()>
[2]: 0x400dfb <non-virtual thunk to C5::norf()>
(gdb)
```

Εικόνα 18. obj6 vtables

Γενικά παρατηρήσαμε ότι δεν εισήχθησαν offsets σε κάποιο από τα vtable, με σκοπό να συνδυαστούν μεταξύ τους. Η μορφή των αντικειμένων στην μνήμη είναι αρκιβώς αυτή που είδαμε και στο μάθημα. Ακόμη είδαμε τις “thunk” συναρτήσεις, που είναι απαραίτητες για την ρύθμιση του this.