V-tables & Multiple Inheritance
C++, x86-64 architecture

Multiple Inheritance in Constructors

- Class Phone with function makeCall()
- Class Electronic with functions turnOn(), turnOff()
- Class CellPhone inherits from both

```
#ifndef CELLPHONE_H
#define CELLPHONE_H

#include "Phone.h"
#include "Electronic.h"
#include <string>

class CellPhone : public Phone, public Electronic
{
public:
CellPhone() : m_isOn(false) {}
~CellPhone() {}

void turnOn();
void turnOff();
void makeCall();

private:
bool m_isOn;
};

#endif
```

Main

```
int main(int argc, char* argv[])
{
    CellPhone* cell = new CellPhone();
    cell->turnOn();
    cell->makeCall();
    cell->turnOff();
    Phone* phone = cell;
    phone->makeCall();
    Electronic* elect = cell;
    elect->turnOn();
    elect->turnOff();
    electronicFunc(cell);
    phoneFunc(cell);
    cellFunc(cell);

    return 0;
}
```

```
void electronicFunc(Electronic* e)
{
    e->turnOn();
    e->turnOff();
}

void phoneFunc(Phone* p)
{
    p->makeCall();
}

void cellFunc(CellPhone* c)
{
    c->turnOn();
    c->makeCall();
    c->turnOff();
}
```

Cellphone Constructor

Examining assembly for following line of code:

CellPhone* cell = new CellPhone();

Called in main as follows:

call _ZN9CellPhoneC1Ev

```
# Name of constructor
_ZN9CellPhoneC2Ev:
# save base pointer and update base pointer, stack pointer
pushq   %rbp
movq    %rsp, %rbp
subq$16, %rsp
# store this on stack and move into %rax
movq    %rdi, -8(%rbp)
movq    -8(%rbp), %rax
# pass object into phone constructor
movq    %rax, %rdi
call    _ZN5PhoneC2Ev
# pass object at offset of 8 into electronics constructor
movq    -8(%rbp), %rax
addq    $8, %rax
movq    %rax, %rdi
call    _ZN10ElectronicC2Ev
# set up vptr for phone at 0 and vptr for electronic at 8
movq    -8(%rbp), %rax
movq    $_ZTV9CellPhone+16, (%rax)
movq    -8(%rbp), %rax
movq    $_ZTV9CellPhone+72, 8(%rax)
movq    -8(%rbp), %rax
movb    $0, 16(%rax)    # set m_isOn to 0
leave
ret
```

Cellphone Constructor

- After constructor call, cellphone object has two v-pointers
  - Vptr for phone at offset 0
  - Vptr for electronic at offset 8
- Vptrs followed by member variables (just m_isOn in this case)
- Each vptr points at v-table for specified object type

Upcasting

Examine following part of main:

```
Phone* phone = cell;

phone->makeCall();
```

```
# Move cell into %rax
movq    -24(%rbp), %rax
# Copy that into new phone pointer
movq    %rax, -32(%rbp)
# Get vtable using vptr at offset 0
movq    -32(%rbp), %rax
movq    (%rax), %rax
# Add 16 to get to makeCall() function in vtable
addq $16, %rax
movq    (%rax), %rax
movq    -32(%rbp), %rdx
movq    %rdx, %rdi
# Call function pointer to in %rax with phone as first argument
call   *%rax
```

Upcasting

- Recall from construction that vptr for phone object is at offset 0 in cellphone object
- Therefore, when creating Phone*, no need to add offset, just copy Phone vptr over

Upcasting

- When upcasting cell to an Electronic pointer, the Electronic vptr is found at an offset of 8
- Set this vptr equal to new Electronic pointer

Upcasting

Examine following part of main:

```
Electronic* elect = cell;

elect->turnOn();

elect->turnOff();
```

```
cmpq    $0, -24(%rbp)  # Check if object is null
je .L15 # If so, jump to .L15
movq    -24(%rbp), %rax  # Otherwise, get vptr for electronic
addq    $8, %rax  # must add 8 to get to electronic vptr
jmp  .L16
.L15:
movl$0, %eax # Set eax to 0
.L16:
movq    %rax, -40(%rbp)  # Store value in %rax as elect
movq    -40(%rbp), %rax  # Move elect into %rax and get vtable
movq    (%rax), %rax
addq $16, %rax   # Find turnOn function at offset of 16 in vtable
movq    (%rax), %rax
movq    -40(%rbp), %rdx
movq    %rdx, %rdi
call   *%rax   # Call function pointer to in %rax with elect as argument
movq    -40(%rbp), %rax
movq    (%rax), %rax
addq $24, %rax   # Find turnOff function at offset 24 in vtable
movq    (%rax), %rax
movq    -40(%rbp), %rdx
movq    %rdx, %rdi
call   *%rax   # Call function pointer to in %rax with elect as argument
```