Intro to the C++ Object Model

Richard Powell <rmpowell@me.com>

9/23/2015 - v1.4

Intro to my Intro

- Why this talk
- All code here-in is example code
  - Production code should be code-reviewed, build with no warnings, etc…

ProTip!

`make` is your friend

```
$ cat intro.cpp
#include <iostream>
int main() { std::cout<<"hello world\n"; }

$ make intro
c++   intro.cpp   -o intro

$ ./intro
hello world
```

- `make`'s auto deduction rules will deduce they should use `$CXX` to convert cpp files to executables.

ProTip!

Celebrate like it's 2014

```
$ make CXXFLAGS="-std=c++14 -stdlib=libc++" intro
c++ -std=c++1y -stdlib=libc++   intro.cpp   -o intro

$ ./intro
hello world
```

- '`-std=c++14`' is the flag to use C++14 (or '`-std=c++1y`' on legacy)
- '`-stdlib=libc++`' will use libc++, the recommended library for C++14

Object-Oriented Programming

Object-Oriented Programming

- What is Object-Oriented Programming?

Object-Oriented Programming

- What is Object-Oriented Programming?
- “Object-oriented programming (OOP) is a programming paradigm that represents the concept of "objects" that have data fields (attributes that describe the object) and associated procedures known as methods” - http://en.wikipedia.org/wiki/Object-oriented_programming

Object-Oriented Programming

- What is Object-Oriented Programming?
- “Object-oriented programming (OOP) is a programming paradigm that represents the concept of "objects" that have data fields (attributes that describe the object) and associated procedures known as methods” - http://en.wikipedia.org/wiki/Object-oriented_programming
- “In this way, the data structure becomes an object that includes both data and functions.” - http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html

C++ Object model

C++ Object model

- “An object is a region of storage.” ISO N3690: § 1.8 [intro.object]

C++ Object model

- “An object is a region of storage.” ISO N3690: § 1.8 [intro.object]
- If you were to design a system that has inheritance and run-time determined functionality (polymorphism), how would you do it?

C++ Object model

- “An object is a region of storage.” ISO N3690: § 1.8 [intro.object]
- If you were to design a system that has inheritance and run-time determined functionality (polymorphism), how would you do it?
  - Constraints:

C++ Object model

- “An object is a region of storage.” ISO N3690: § 1.8 [intro.object]
- If you were to design a system that has inheritance and run-time determined functionality (polymorphism), how would you do it?
  - Constraints:
    - Compatiblity with ANSI-C

C++ Object model

- “An object is a region of storage.” ISO N3690: § 1.8 [intro.object]
- If you were to design a system that has inheritance and run-time determined functionality (polymorphism), how would you do it?
  - Constraints:
    - Compatiblity with ANSI-C
    - “You shouldn’t pay for what you don’t use”

quiz_c_size.c:

```
$ make quiz_c_size
$ ./quiz_c_size
sizeof(float): 4
```

```c
#include <stdio.h>

typedef struct
{
  float real;
  float imag;
} Complex;

int main()
{
  printf("sizeof(float): %ld\n", sizeof(float));
  printf("sizeof(Complex): %ld\n", sizeof(Complex));
}
```

A

```
sizeof(Complex): 4
```

B

```
sizeof(Complex): 8
```

C

```
sizeof(Complex): 16
```

quiz_c_size.c:

```
$ make quiz_c_size
$ ./quiz_c_size
sizeof(float): 4
```

```c
#include <stdio.h>

typedef struct
{
  float real;
  float imag;
} Complex;

int main()
{
  printf("sizeof(float): %ld\n", sizeof(float));
  printf("sizeof(Complex): %ld\n", sizeof(Complex));
}
```

A

```
sizeof(Complex): 4
```

B

```
sizeof(Complex): 8
```

C

```
sizeof(Complex): 16
```

quiz_c_offset.c:

```
#include <stdio.h>

typedef struct
{
  float real;
  float imag;
} Complex;

int main()
{
  Complex c;
  printf("address of c: %p\n", &c);
  printf("address of c.real: %p\n", &c.real);
  printf("address of c.imag: %p\n", &c.imag);
}
```

```
$ make quiz_c_offset
$ ./quiz_c_offset
address of c: 0x10080
```

A

```
address of c.real: 0x10080
address of c.imag: 0x10084
```

B

```
address of c.real: 0x10084
address of c.imag: 0x10080
```

C

```
address of c.real: 0x10084
address of c.imag: 0x10088
```

quiz_c_offset.c:

```c
#include <stdio.h>

typedef struct
{
  float real;
  float imag;
} Complex;

int main()
{
  Complex c;
  printf("address of c: %p\n", &c);
  printf("address of c.real: %p\n", &c.real);
  printf("address of c.imag: %p\n", &c.imag);
}
```

```
$ make quiz_c_offset
$ ./quiz_c_offset
address of c: 0x10080
```

A

```
address of c.real: 0x10080
address of c.imag: 0x10084
```

B

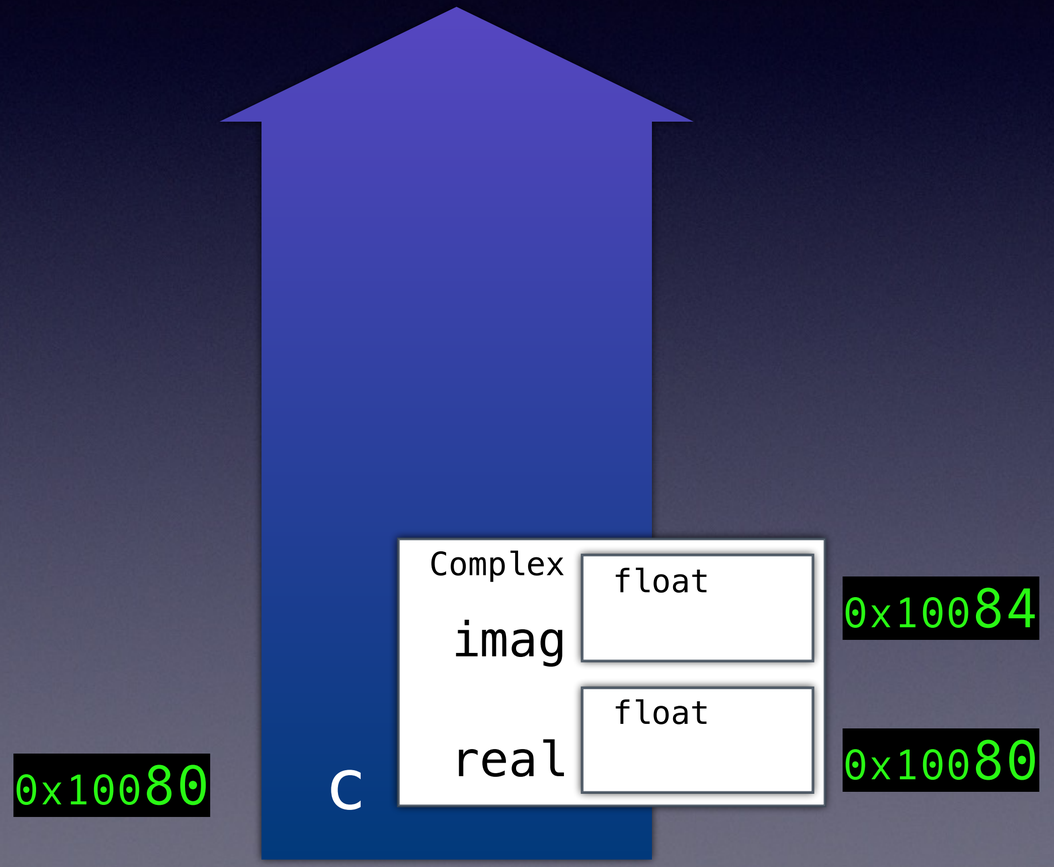
```
address of c.real: 0x10084
address of c.imag: 0x10080
```

C

```
address of c.real: 0x10084
address of c.imag: 0x10088
```

quiz_c_offset.c:

```c
#include <stdio.h>

typedef struct
{
  float real;
  float imag;
} Complex;

int main()
{
  Complex c;
  printf("address of c: %p\n", &c);
  printf("address of c.real: %p\n", &c.real);
  printf("address of c.imag: %p\n", &c.imag);
}
```



quiz_size1.cpp:

```
$ make quiz_size1
$ ./quiz_size1
sizeof(float): 4
```

```cpp
#include <iostream>

struct Complex
{
  float real;
  float imag;
};

int main()
{
  std::cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  std::cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
}
```

A

```
sizeof(Complex): 4
```

B

```
sizeof(Complex): 8
```

C

```
sizeof(Complex): 16
```

quiz_size1.cpp:

```
$ make quiz_size1
$ ./quiz_size1
sizeof(float): 4
```

```cpp
#include <iostream>

struct Complex
{
  float real;
  float imag;
};

int main()
{
  std::cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  std::cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
}
```

A `sizeof(Complex): 4`

B `sizeof(Complex): 8`

C `sizeof(Complex): 16`

quiz_offset1.cpp:

```
#include <iostream>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

int main()
{
  Complex c;
  cout<<"address of c: "<<&c<<"\n";
  cout<<"address of c.real: "<<&c.real<<"\n";
  cout<<"address of c.imag: "<<&c.imag<<"\n";
}
```

```
$ make quiz_offset1
$ ./quiz_offset1
address of c: 0x10080
```

A

```
address of c.real: 0x10080
address of c.imag: 0x10084
```

B

```
address of c.real: 0x10084
address of c.imag: 0x10080
```

C

```
address of c.real: 0x10084
address of c.imag: 0x10088
```

ProTip!

```
#include <iostream>
using std::cout;
```

- The `using` directive controls importing of symbols from namespaces
- You *can* import all symbols from a namespace (i.e. “`using namespace std`”), but “Good” programmers are selective and surgical
- Only import what you need
- NEVER blindly use `using` to import symbols into a header file!

quiz_offset1.cpp:

```
#include <iostream>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

int main()
{
  Complex c;
  cout<<"address of c: "<<&c<<"\n";
  cout<<"address of c.real: "<<&c.real<<"\n";
  cout<<"address of c.imag: "<<&c.imag<<"\n";
}
```

```
$ make quiz_offset1
$ ./quiz_offset1
address of c: 0x10080
```

A

```
address of c.real: 0x10080
address of c.imag: 0x10084
```

B

```
address of c.real: 0x10084
address of c.imag: 0x10080
```

C

```
address of c.real: 0x10084
address of c.imag: 0x10088
```

quiz_offset1.cpp:

```
#include <iostream>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

int main()
{
  Complex c;
  cout<<"address of c: "<<&c<<"\n";
  cout<<"address of c.real: "<<&c.real<<"\n";
  cout<<"address of c.imag: "<<&c.imag<<"\n";
}
```

```
$ make quiz_offset1
$ ./quiz_offset1
address of c: 0x10080
```

A

```
address of c.real: 0x10080
address of c.imag: 0x10084
```

B

```
address of c.real: 0x10084
address of c.imag: 0x10080
```

C

```
address of c.real: 0x10084
address of c.imag: 0x10088
```

POD

- C++ rule is that member variables declared later in a structure must be at a higher address
- These are called POD objects (Plain Old Data)
- Objects of POD types are fully compatible with the C programming language.

ProTip!

is_pod

demo_is_pod.cpp:

```
#include <iostream>
#include <type_traits>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

int main()
{
  cout<<"is Complex a POD? "<<
    (std::is_pod<Complex>() ? "yes" : "no")<<"\n";
}
```

```
$ make demo_is_pod
$ ./demo_is_pod
is Complex a POD? yes
```

quiz_size_class.cpp:

```
$ make quiz_size_class
$ ./quiz_size_class
sizeof(float): 4
```

```cpp
#include <iostream>
using std::cout;

class Complex
{
  float real;
  float imag;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
}
```

A

```
sizeof(Complex): 4
```

B

```
sizeof(Complex): 8
```

C

```
sizeof(Complex): 16
```

quiz_size_class.cpp:

```
$ make quiz_size_class
$ ./quiz_size_class
sizeof(float): 4
```

```cpp
#include <iostream>
using std::cout;

class Complex
{
  float real;
  float imag;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
}
```

A

```
sizeof(Complex): 4
```

B

```
sizeof(Complex): 8
```

C

```
sizeof(Complex): 16
```

- In C++, the only difference between a struct and a class is the default accessor value

```
class Complex
{
  float real;
  float imag;
};
```

==

```
struct Complex
{
private:
  float real;
  float imag;
};
```

quiz_derived.cpp:

```
#include <iostream>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
  cout<<"sizeof(Derived): "<<sizeof(Derived)<<"\n";
}
```

```
$ make quiz_derived
$ ./quiz_derived
sizeof(float): 4
```

A

```
sizeof(Complex): 8
sizeof(Derived): 8
```

B

```
sizeof(Complex): 8
sizeof(Derived): 12
```

C

```
sizeof(Complex): 8
sizeof(Derived): 16
```

quiz_derived.cpp:

```
#include <iostream>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
  cout<<"sizeof(Derived): "<<sizeof(Derived)<<"\n";
}
```

```
$ make quiz_derived
$ ./quiz_derived
sizeof(float): 4
```

A

```
sizeof(Complex): 8
sizeof(Derived): 8
```

B

```
sizeof(Complex): 8
sizeof(Derived): 12
```

C

```
sizeof(Complex): 8
sizeof(Derived): 16
```

quiz_offset2.cpp:

```
#include <iostream>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};

int main()
{
  Derived d;
  cout<<"address of d: "<<(&d)<<"\n";
  cout<<"address of d.real: "<<(&d.real)<<"\n";
  cout<<"address of d.imag: "<<(&d.imag)<<"\n";
  cout<<"address of d.angle: "<<(&d.angle)<<"\n";
}
```

```
$ make quiz_offset2
$ ./quiz_offset2
address of d: 0x10080
```

A

```
address of d.real: 0x10080
address of d.imag: 0x10084
address of d.angle: 0x10088
```

B

```
address of d.real: 0x10084
address of d.imag: 0x10088
address of d.angle: 0x10080
```

C

```
address of d.real: 0x10084
address of d.imag: 0x10088
address of d.angle: 0x1008c
```

quiz_offset2.cpp:

```
#include <iostream>
using std::cout;

struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};

int main()
{
  Derived d;
  cout<<"address of d: "<<(&d)<<"\n";
  cout<<"address of d.real: "<<(&d.real)<<"\n";
  cout<<"address of d.imag: "<<(&d.imag)<<"\n";
  cout<<"address of d.angle: "<<(&d.angle)<<"\n";
}
```

```
$ make quiz_offset2
$ ./quiz_offset2
address of d: 0x10080
```

A

```
address of d.real: 0x10080
address of d.imag: 0x10084
address of d.angle: 0x10088
```

B

```
address of d.real: 0x10084
address of d.imag: 0x10088
address of d.angle: 0x10080
```

C

```
address of d.real: 0x10084
address of d.imag: 0x10088
address of d.angle: 0x1008c
```

- Inheritance works by “extending” the object
- Think of inheritance as essentially “stacking” subclasses on top of base classes

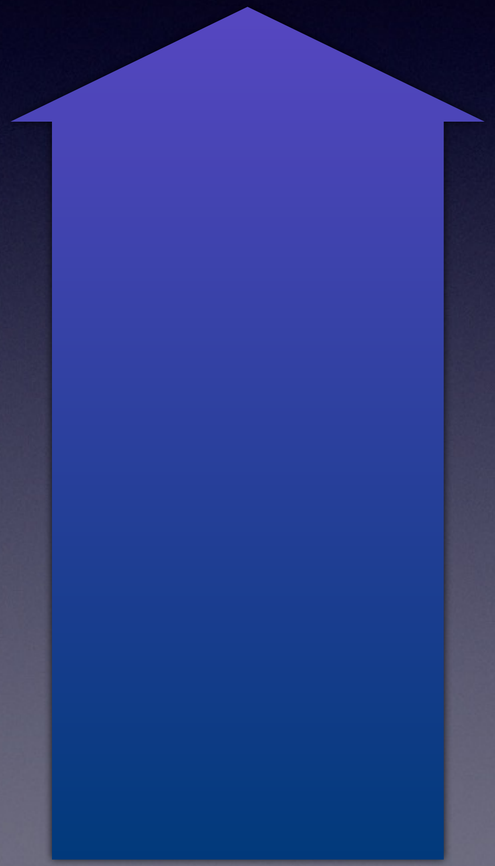
```
struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};
```

“==“

```
struct Derived
{
  struct {
    float real;
    float imag;
  };
  float angle;
};
```

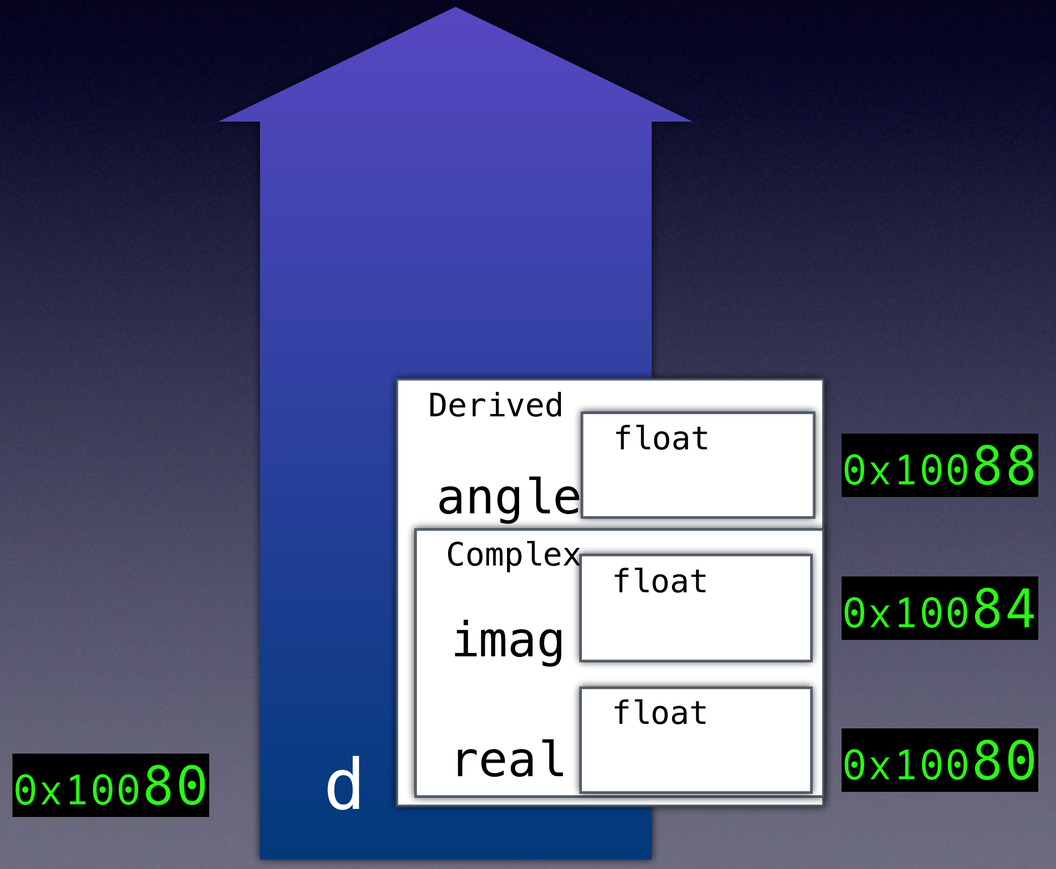
Compiler's view of the stack

```
#include <iostream>

struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};

int main()
{
  Derived d;
  Complex& c = d;
}
```



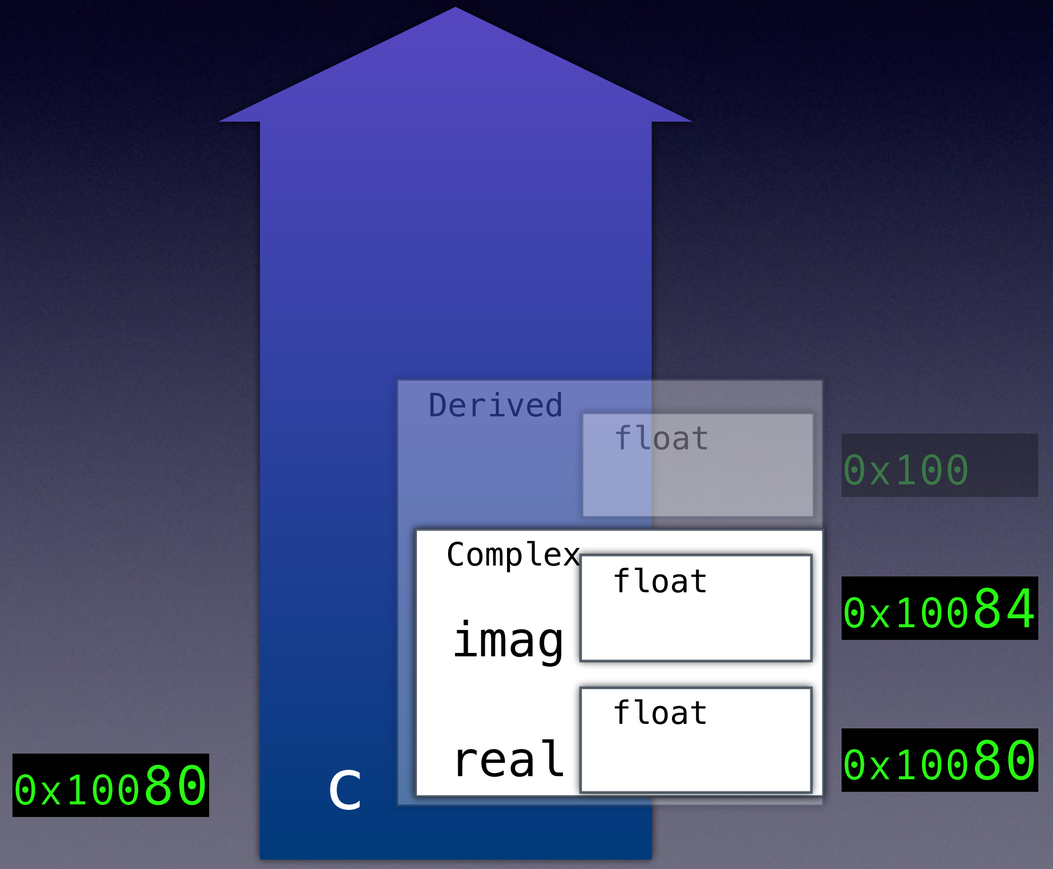
Compiler's view of the stack
*What **d** looks like:*

```
#include <iostream>

struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};

int main()
{
  Derived d;
  Complex& c = d;
}
```



Compiler's view of the stack
What **c** *looks like:*

```
#include <iostream>

struct Complex
{
  float real;
  float imag;
};

struct Derived : public Complex
{
  float angle;
};

int main()
{
  Derived d;
  Complex& c = d;
}
```



quiz_mem_func.cpp:

```
$ make quiz_mem_func
$ ./quiz_mem_func
sizeof(float): 4
```

```cpp
#include <iostream>
#include <cmath>
using std::cout;

struct Complex
{
  float Abs() const { return std::hypot(real, imag); }
  float real;
  float imag;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
}
```

A

```
sizeof(Complex): 4
```

B

```
sizeof(Complex): 8
```

C

```
sizeof(Complex): 16
```

quiz_mem_func.cpp:

```
$ make quiz_mem_func
$ ./quiz_mem_func
sizeof(float): 4
```

```cpp
#include <iostream>
#include <cmath>
using std::cout;

struct Complex
{
  float Abs() const { return std::hypot(real, imag); }
  float real;
  float imag;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
}
```

A `sizeof(Complex): 4`

B `sizeof(Complex): 8`

C `sizeof(Complex): 16`

Conceptually how member function are formed

```
struct Complex
{
  float Abs() const;
  float real;
  float imag;
};
```

```
float Complex::Abs() const
{
  return std::hypot(real, imag);
}
```

```
Complex c;
auto v = c.Abs();
```

```
float Complex::Abs() const
{
  return std::hypot(real, imag);
}
```

```
Complex c;
auto v = c.Abs();
```

A pointer the `this` object is added as the first argument

```
float Complex::Abs() const
{
  return std::hypot(real, imag);
}
```

```
Complex c;
auto v = c.Abs();
```

A pointer the `this` object is added as the first argument

```
float Complex::Abs(Complex * this) const
{
  return std::hypot(real, imag);
}
```

```
Complex c;
auto v = c.Abs();
```

The CV-qualifiers are applied to the `this` object

```
float Complex::Abs(Complex * this) const
{
  return std::hypot(real, imag);
}
```

```
Complex c;
auto v = c.Abs();
```

The CV-qualifiers are applied to the `this` object

```
float Complex::Abs(Complex const * this)
{
  return std::hypot(real, imag);
}
```

```
Complex c;
auto v = c.Abs();
```

The CV-qualifiers are applied to the `this` object

```
float Complex::Abs(Complex const * this)
{
  return std::hypot(real, imag);
}
```

For function invocation, the object is moved to first argument

```
Complex c;
auto v = c.Abs();
```

The CV-qualifiers are applied to the `this` object

```
float Complex::Abs(Complex const * this)
{
  return std::hypot(real, imag);
}
```

For function invocation, the object is moved to first argument

```
Complex c;
auto v = Abs(&c);
```

All member variables are prefixed to use the `this` object

```
float Complex::Abs(Complex const * this)
{
  return std::hypot(real, imag);
}
```

```
Complex c;
auto v = Abs(&c);
```

All member variables are prefixed to use the `this` object

```
float Complex::Abs(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

```
Complex c;
auto v = Abs(&c);
```

The compiler translates the function to a free function with a implementation defined “name mangled” version

```
float Complex::Abs(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

```
Complex c;
auto v = Abs(&c);
```

The compiler translates the function to a free function with a implementation defined “name mangled” version

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

For function invocation, the function call is also translated

```
Complex c;
auto v = Abs(&c);
```

The compiler translates the function to a free function with a implementation defined “name mangled” version

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

For function invocation, the function call is also translated

```
Complex c;
auto v = __ZNK7Complex3AbsEv(&c);
```

Before

```
float Complex::Abs() const
{
  return std::hypot(real, imag);
}
```

After

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

ProTip!

`c++filt` to translate

```
$ c++filt __ZNK7Complex3AbsEv
Complex::Abs() const
```

- Use `c++filt` to translate name mangled functions to original names

```cpp
#include <cmath>

struct Complex
{
  float Abs() const
  {
    return std::hypot(real, imag);
  }
  float real;
  float imag;
};
```

“==“

```cpp
#include <math.h>

typedef struct
{
  float real;
  float imag;
} Complex;

float __ZNK7Complex3AbsEv(Complex const* this)
{
  return hypot(this->real, this->imag);
}
```

```cpp
#include <iostream>
using std::cout;

struct Erdos
{
  void whoAmI() { cout<<"I am Erdos\n"; }
  virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
  void whoAmI() { cout<<"I am Fermat\n"; }
  virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};
```

quiz_virtual1.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   Erdos e;
   e.whoAmI();
   e.whoAmIReally();

   Fermat f;
   f.whoAmI();
   f.whoAmIReally();
}
```

```
$ make quiz_virtual1
$ ./quiz_virtual1
I am Erdos
I really am Erdos
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual1.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   Erdos e;
   e.whoAmI();
   e.whoAmIReally();

   Fermat f;
   f.whoAmI();
   f.whoAmIReally();
}
```

```
$ make quiz_virtual1
$ ./quiz_virtual1
I am Erdos
I really am Erdos
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual2.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   Fermat f;
   f.whoAmI();
   f.whoAmIReally();

   Erdos& e = f;
   e.whoAmI();
   e.whoAmIReally();
}
```

```
$ make quiz_virtual2
$ ./quiz_virtual2
I am Fermat
I really am Fermat
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual2.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   Fermat f;
   f.whoAmI();
   f.whoAmIReally();

   Erdos& e = f;
   e.whoAmI();
   e.whoAmIReally();
}
```

```
$ make quiz_virtual2
$ ./quiz_virtual2
I am Fermat
I really am Fermat
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

```
#include <iostream>
using std::cout;

struct Erdos
{
  void whoAmI() { cout<<"I am Erdos\n"; }
  virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
  void whoAmI() { cout<<"I am Fermat\n"; }
  virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};
```

- Non-virtual member functions bind statically. Function resolution occurs at compile time.

- Virtual member functions bind dynamically. Function resolution occurs when the object is created.

quiz_virtual2.cpp:

```
$ make quiz_virtual2
$ ./quiz_virtual2
I am Fermat
I really am Fermat
```

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   Fermat f;
   f.whoAmI();
   f.whoAmIReally();

   Erdos& e = f;
   e.whoAmI();
   e.whoAmIReally();
}
```

non-virtual functions resolved statically

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual2.cpp:

```
$ make quiz_virtual2
$ ./quiz_virtual2
I am Fermat
I really am Fermat
```

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   Fermat f;
   f.whoAmI();
   f.whoAmIReally();

   Erdos& e = f;
   e.whoAmI();
   e.whoAmIReally();
}
```

Virtual calls determined by object creation.
Original object created as Fermat.
Fermat function is called.

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual3.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   Erdos * e1 = new Erdos;
   e1->whoAmI();
   e1->whoAmIReally();

   Erdos * e2 = new Fermat;
   e2->whoAmI();
   e2->whoAmIReally();
}
```

```
$ make quiz_virtual3
$ ./quiz_virtual3
I am Erdos
I really am Erdos
```

A

```
I am Erdos
I really am Erdos
```

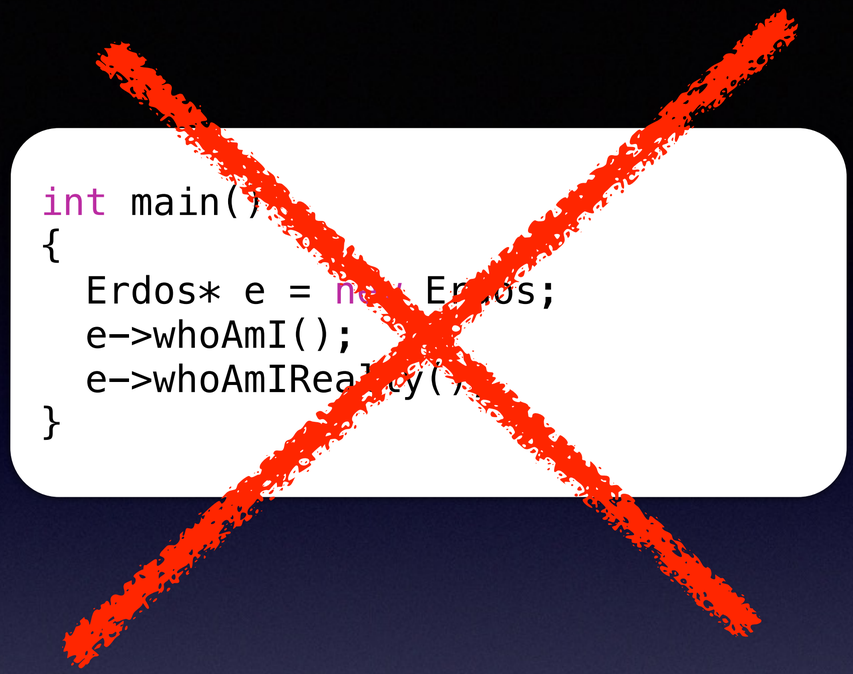
B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

```
int main()
{
  Erdos* e = new Erdos;
  e->whoAmI();
  e->whoAmIReally();
}
```


int main()
{
Erdos* e = new Erdos;
e->whoAmI();
e->whoAmIReally();
}

ProTip!

```
int main()
{
  Erdos* e = new Erdos;
  e->whoAmI();
  e->whoAmIReally();
}
```

```
int main()
{
  using std::unique_ptr;
  using std::make_unique;

  unique_ptr<Erdos> e = make_unique<Erdos>();
  e->whoAmI();
  e->whoAmIReally();
}
```

ProTip!

```
int main()
{
  Erdos* e = new Erdos;
  e->whoAmI();
  e->whoAmIReally();
}
```

```
int main()
{
  using std::unique_ptr;
  using std::make_unique;

  unique_ptr<Erdos> e = make_unique<Erdos>();
  e->whoAmI();
  e->whoAmIReally();
}
```

- `unique_ptr` is a C++ Smart Pointer
- `make_unique`: convenience function to make `unique_ptr`s
- manages raw resources
- ALWAYS use smart pointers
- NEVER write new in your code

quiz_virtual3.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   using std::unique_ptr;
   using std::make_unique;

   unique_ptr<Erdos> e1 = make_unique<Erdos>();
   e1->whoAmI();
   e1->whoAmIReally();

   unique_ptr<Erdos> e2 = make_unique<Fermat>();
   e2->whoAmI();
   e2->whoAmIReally();
}
```

```
$ make quiz_virtual3
$ ./quiz_virtual3
I am Erdos
I really am Erdos
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual3.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   using std::unique_ptr;
   using std::make_unique;

   unique_ptr<Erdos> e1 = make_unique<Erdos>();
   e1->whoAmI();
   e1->whoAmIReally();

   unique_ptr<Erdos> e2 = make_unique<Fermat>();
   e2->whoAmI();
   e2->whoAmIReally();
}
```

```
$ make quiz_virtual3
$ ./quiz_virtual3
I am Erdos
I really am Erdos
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual4.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   using std::unique_ptr;
   using std::make_unique;

   unique_ptr<Erdos> e = make_unique<Erdos>();
   e->whoAmI();
   e->whoAmIReally();

   unique_ptr<Fermat> f = make_unique<Fermat>();
   f->whoAmI();
   f->whoAmIReally();
}
```

```
$ make quiz_virtual4
$ ./quiz_virtual4
I am Erdos
I really am Erdos
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_virtual4.cpp:

```
struct Erdos
{
   void whoAmI() { cout<<"I am Erdos\n"; }
   virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
   void whoAmI() { cout<<"I am Fermat\n"; }
   virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
   using std::unique_ptr;
   using std::make_unique;

   unique_ptr<Erdos> e = make_unique<Erdos>();
   e->whoAmI();
   e->whoAmIReally();

   unique_ptr<Fermat> f = make_unique<Fermat>();
   f->whoAmI();
   f->whoAmIReally();
}
```

```
$ make quiz_virtual4
$ ./quiz_virtual4
I am Erdos
I really am Erdos
```

A

```
I am Erdos
I really am Erdos
```

B

```
I am Erdos
I really am Fermat
```

C

```
I am Fermat
I really am Fermat
```

quiz_size3.cpp:

```
struct Complex
{
  virtual ~Complex() = default;
  virtual float Abs() { return std::hypot(real, imag); }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() = default;
  virtual float Abs() { return std::hypot(std::hypot(real, imag), angle); }
  float angle;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(void*): "<<sizeof(void*)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
  cout<<"sizeof(Derived): "<<sizeof(Derived)<<"\n";
}
```

```
$ make quiz_size3
$ ./quiz_size3
sizeof(float): 4
sizeof(void*): 8
```

A

```
sizeof(Complex): 8
sizeof(Derived): 12
```

B

```
sizeof(Complex): 16
sizeof(Derived): 24
```

C

```
sizeof(Complex): 16
sizeof(Derived): 32
```

quiz_size3.cpp:

```
struct Complex
{
  virtual ~Complex() = default;
  virtual float Abs() { return std::hypot(real, imag); }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() = default;
  virtual float Abs() { return std::hypot(std::hypot(real, imag), angle); }
  float angle;
};

int main()
{
  cout<<"sizeof(float): "<<sizeof(float)<<"\n";
  cout<<"sizeof(void*): "<<sizeof(void*)<<"\n";
  cout<<"sizeof(Complex): "<<sizeof(Complex)<<"\n";
  cout<<"sizeof(Derived): "<<sizeof(Derived)<<"\n";
}
```

```
$ make quiz_size3
$ ./quiz_size3
sizeof(float): 4
sizeof(void*): 8
```

A

```
sizeof(Complex): 8
sizeof(Derived): 12
```

B

```
sizeof(Complex): 16
sizeof(Derived): 24
```

C

```
sizeof(Complex): 16
sizeof(Derived): 32
```

quiz_offset3.cpp:

```
struct Complex
{
  virtual ~Complex() = default;
  virtual float Abs() { return std::hypot(real, imag); }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() = default;
  virtual float Abs() { return std::hypot(std::hypot(real, imag), angle); }
  float angle;
};

int main()
{
  Derived d;
  cout<<"address of d: "<<(&d)<<"\n";
  cout<<"address of d.real: "<<(&d.real)<<"\n";
  cout<<"address of d.imag: "<<(&d.imag)<<"\n";
  cout<<"address of d.angle: "<<(&d.angle)<<"\n";
}
```

```
$ make quiz_offset3
$ ./quiz_offset3
address of d: 0x10080
```

A

```
address of d.real: 0x10080
address of d.imag: 0x10084
address of d.angle: 0x10088
```

B

```
address of d.real: 0x10088
address of d.imag: 0x1008C
address of d.angle: 0x10098
```

C

```
address of d.real: 0x10088
address of d.imag: 0x1008C
address of d.angle: 0x10090
```

quiz_offset3.cpp:

```
struct Complex
{
  virtual ~Complex() = default;
  virtual float Abs() { return std::hypot(real, imag); }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() = default;
  virtual float Abs() { return std::hypot(std::hypot(real, imag), angle); }
  float angle;
};

int main()
{
  Derived d;
  cout<<"address of d: "<<(&d)<<"\n";
  cout<<"address of d.real: "<<(&d.real)<<"\n";
  cout<<"address of d.imag: "<<(&d.imag)<<"\n";
  cout<<"address of d.angle: "<<(&d.angle)<<"\n";
}
```

```
$ make quiz_offset3
$ ./quiz_offset3
address of d: 0x10080
```

A

```
address of d.real: 0x10080
address of d.imag: 0x10084
address of d.angle: 0x10088
```

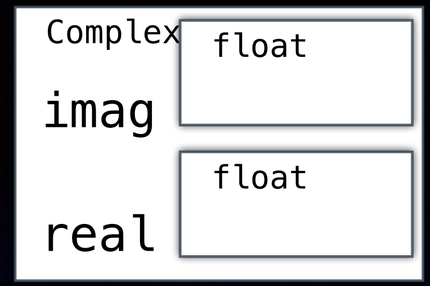
B

```
address of d.real: 0x10088
address of d.imag: 0x1008C
address of d.angle: 0x10098
```

C

```
address of d.real: 0x10088
address of d.imag: 0x1008C
address of d.angle: 0x10090
```

How member function work with virtual

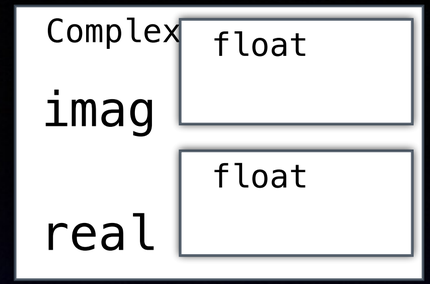



```
struct Complex {
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
float Complex::Abs()
{
  return std::hypot(real, imag);
}
```

```
Complex original_c;
Complex& c = original_c;
float ans = c.Abs();
```

Member functions are generated as normal

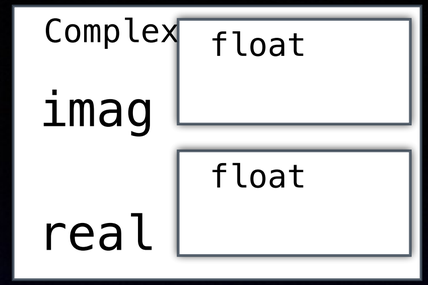



```
struct Complex {
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

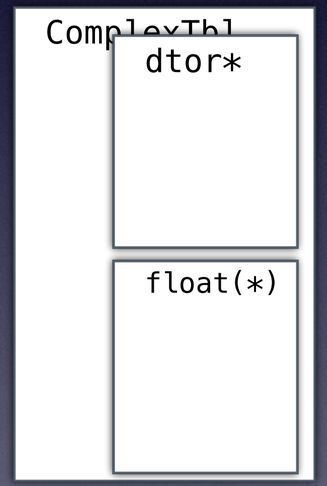
```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

```
Complex original_c;
Complex& c = original_c;
float ans = c.Abs();
```

A table of all virtual functions is generated (the vtable)




```
struct Complex {
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```



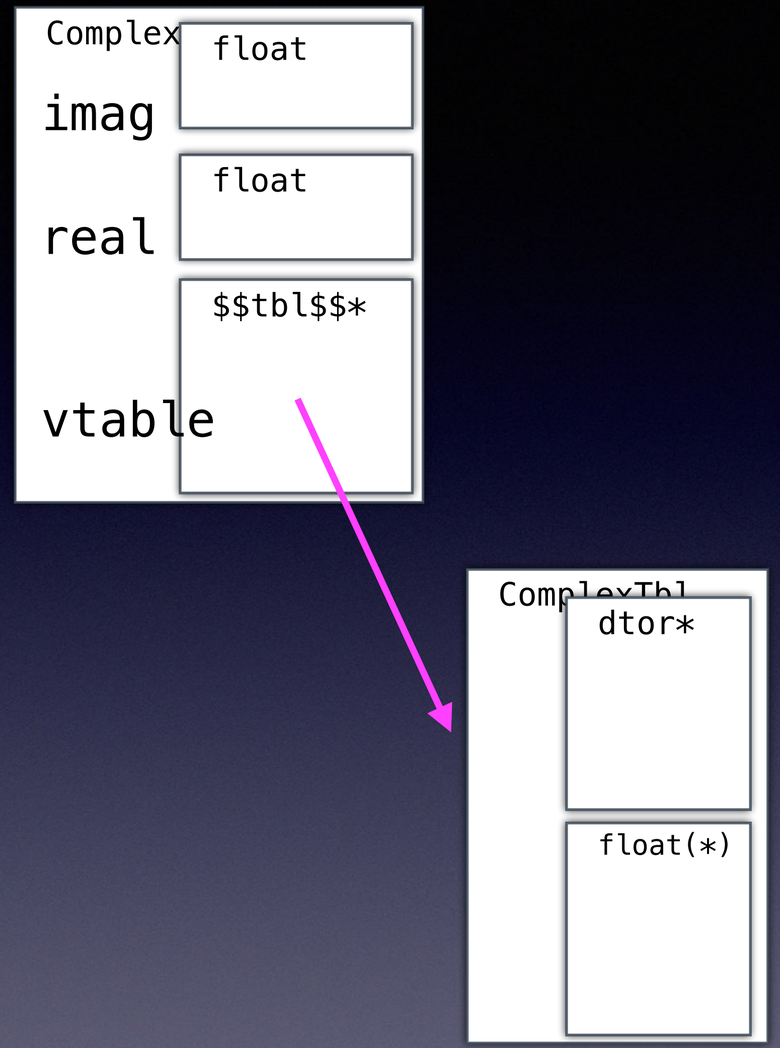

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

```
Complex original_c;
Complex& c = original_c;
float ans = c.Abs();
```

The compiler silently inserts a point to the table

```
struct Complex {
  $$tbl$$ * vtable;
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

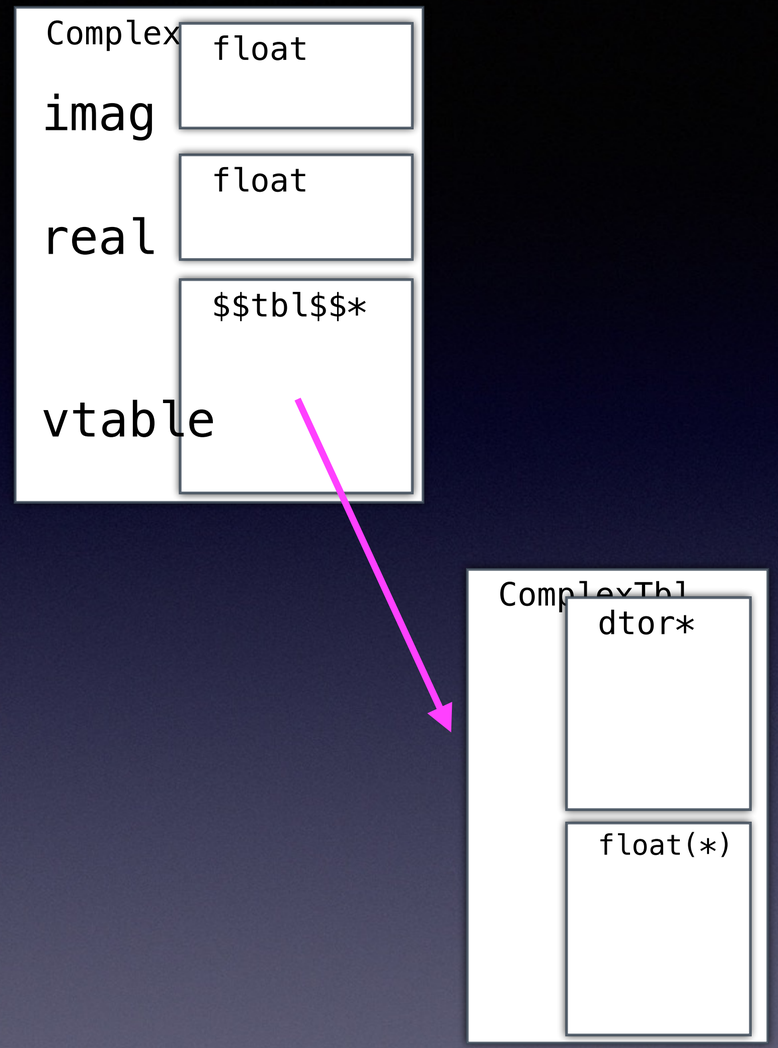
```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```




```
Complex original_c;
Complex& c = original_c;
float ans = c.Abs();
```

```
struct Complex {
  $$tbl$$ * vtable;
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

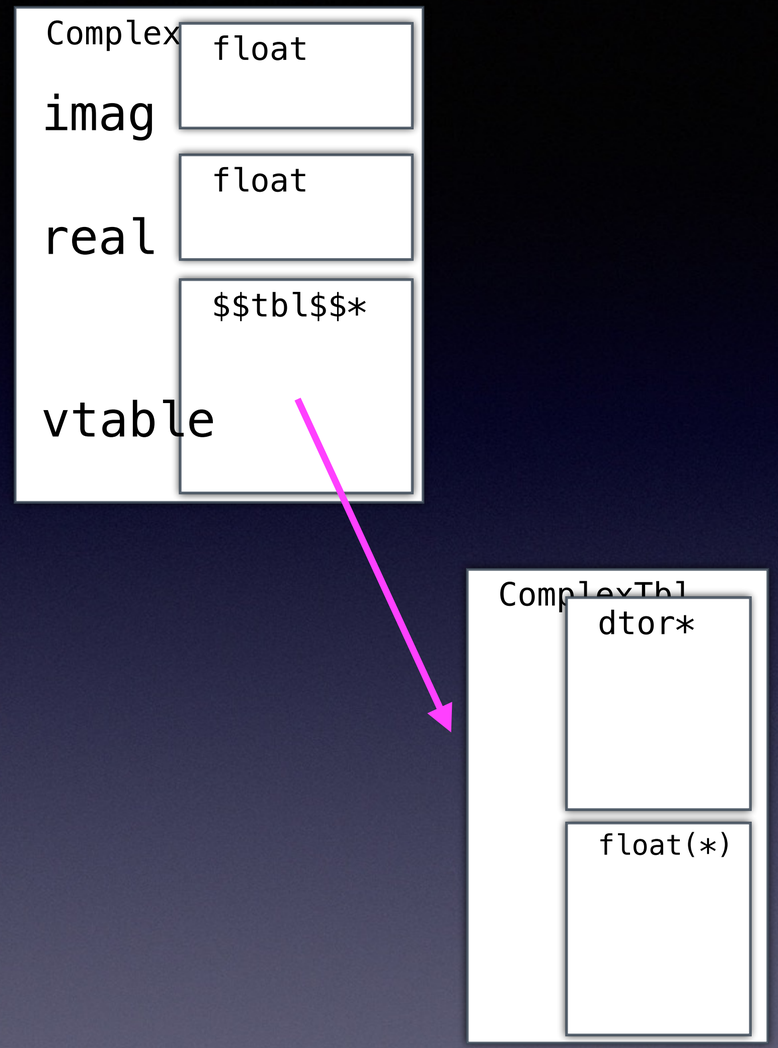


Compiler translates function call into an indirect table lookup and call

```
Complex original_c;
Complex& c = original_c;
float ans = c.Abs();
```

```
struct Complex {
  $$tbl$$ * vtable;
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

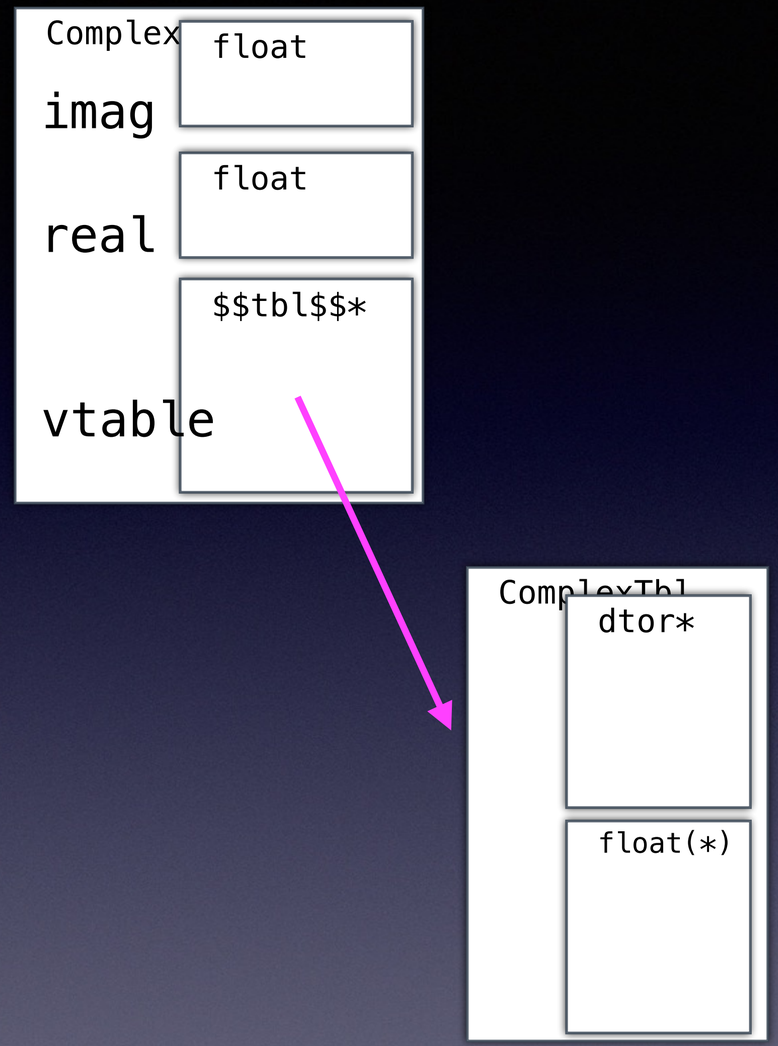


Compiler translates function call into an indirect table lookup and call

```
Complex original_c;
Complex& c = original_c;
float ans = c.vtable[???]();
```

```
struct Complex {
  $$tbl$$ * vtable;
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

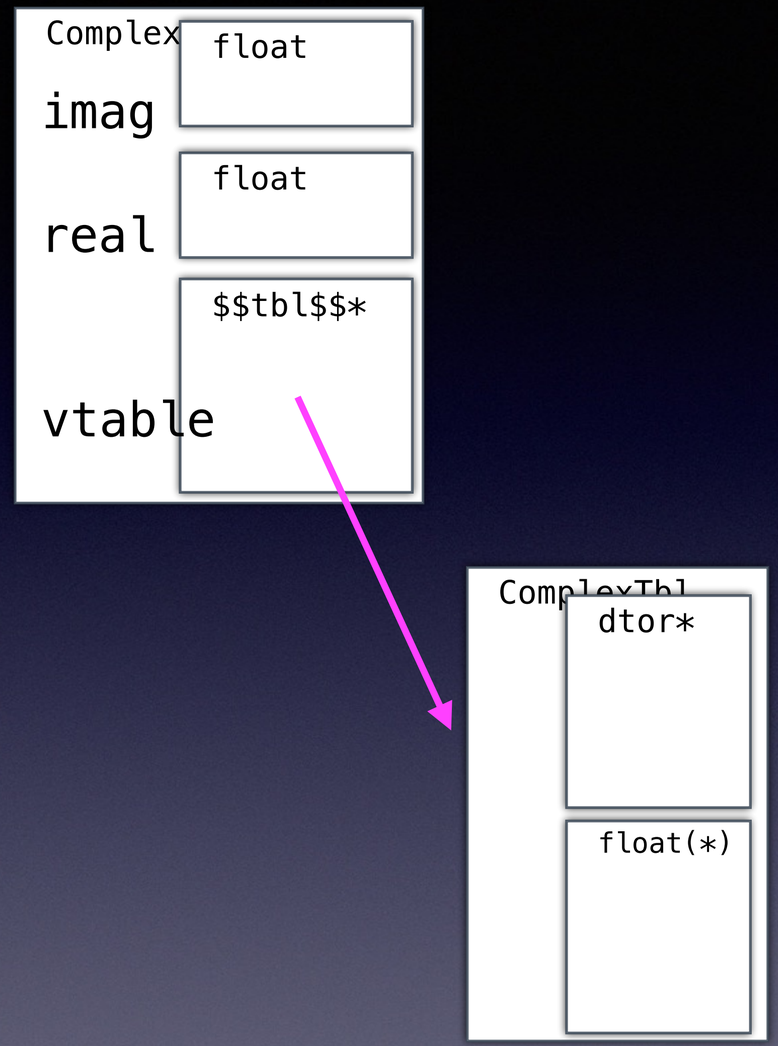


The function offset is determined

```
Complex original_c;
Complex& c = original_c;
float ans = c.vtable[???]();
```

```
struct Complex {
  $$tbl$$ * vtable;
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

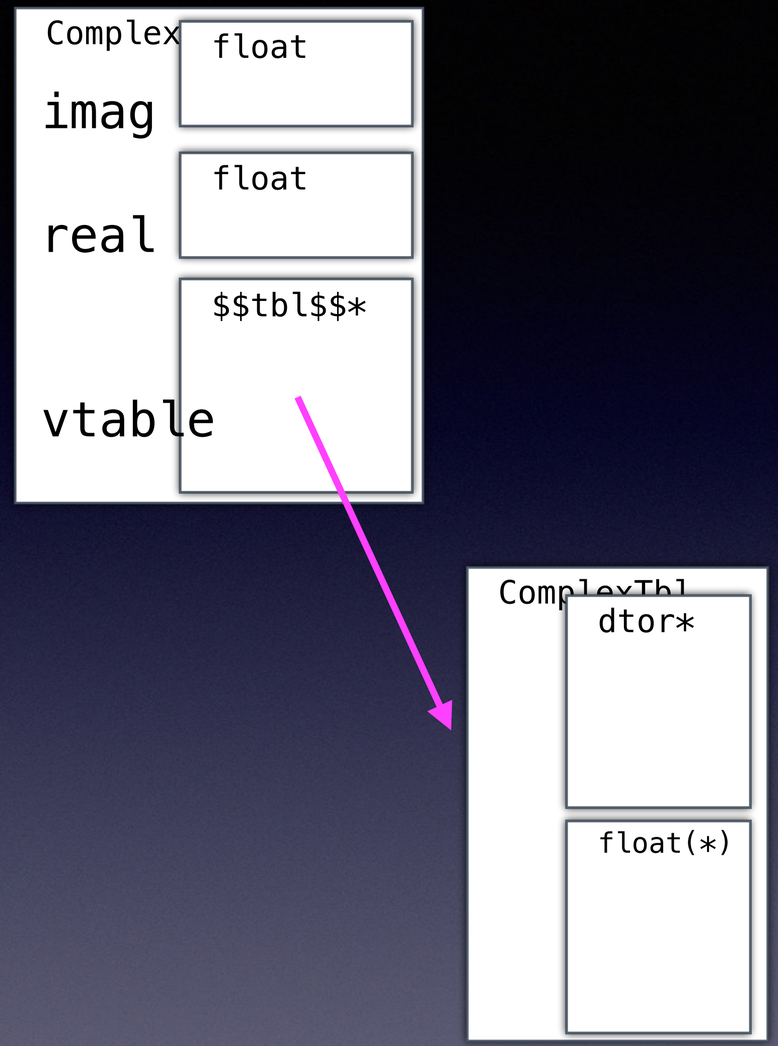


The function offset is determined

```
Complex original_c;
Complex& c = original_c;
float ans = c.vtable[1/*OffsetOf_Abs*/]();
```

```
struct Complex {
  $$tbl$$ * vtable;
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```

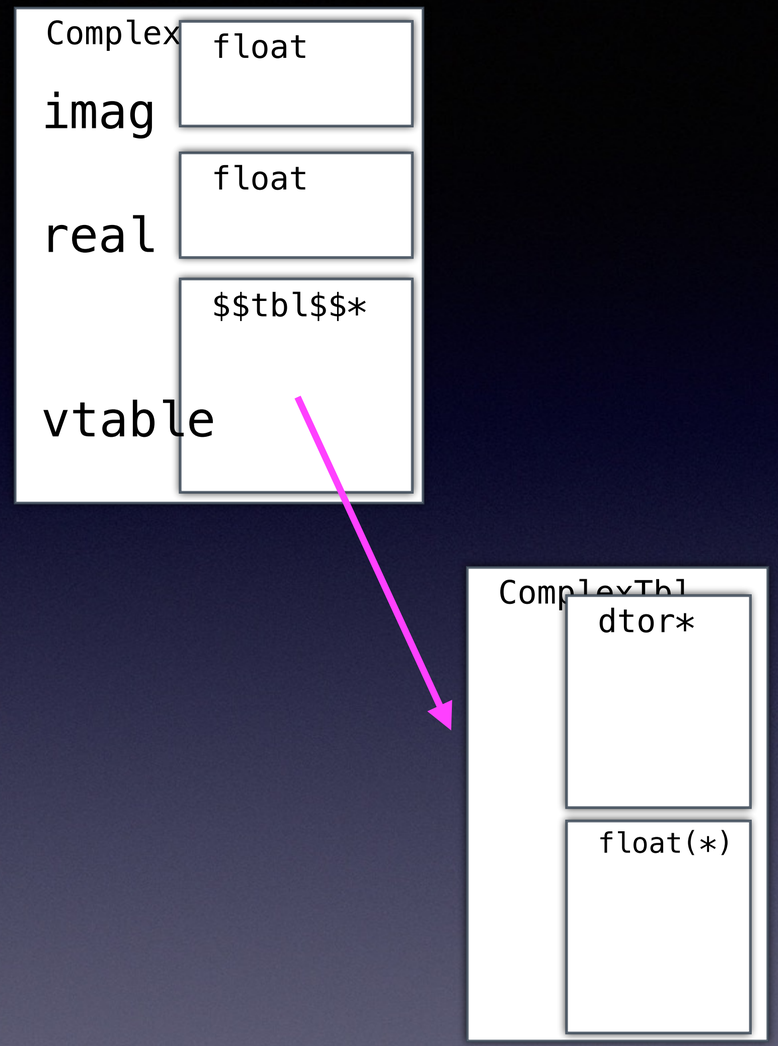


And we pass a pointer to the `this` object

```
Complex original_c;
Complex& c = original_c;
float ans = c.vtable[1/*OffsetOf_Abs*/]();
```

```
struct Complex {
  $$tbl$$ * vtable;
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
float __ZNK7Complex3AbsEv(Complex const * this)
{
  return std::hypot(this->real, this->imag);
}
```



And we pass a pointer to the `this` object

```
Complex original_c;
Complex& c = original_c;
float ans = c.vtable[1/*OffsetOf_Abs*/](&c);
```

```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};
```

```
int main()
{
  Complex d;
}
```

_const

_text

```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};
```

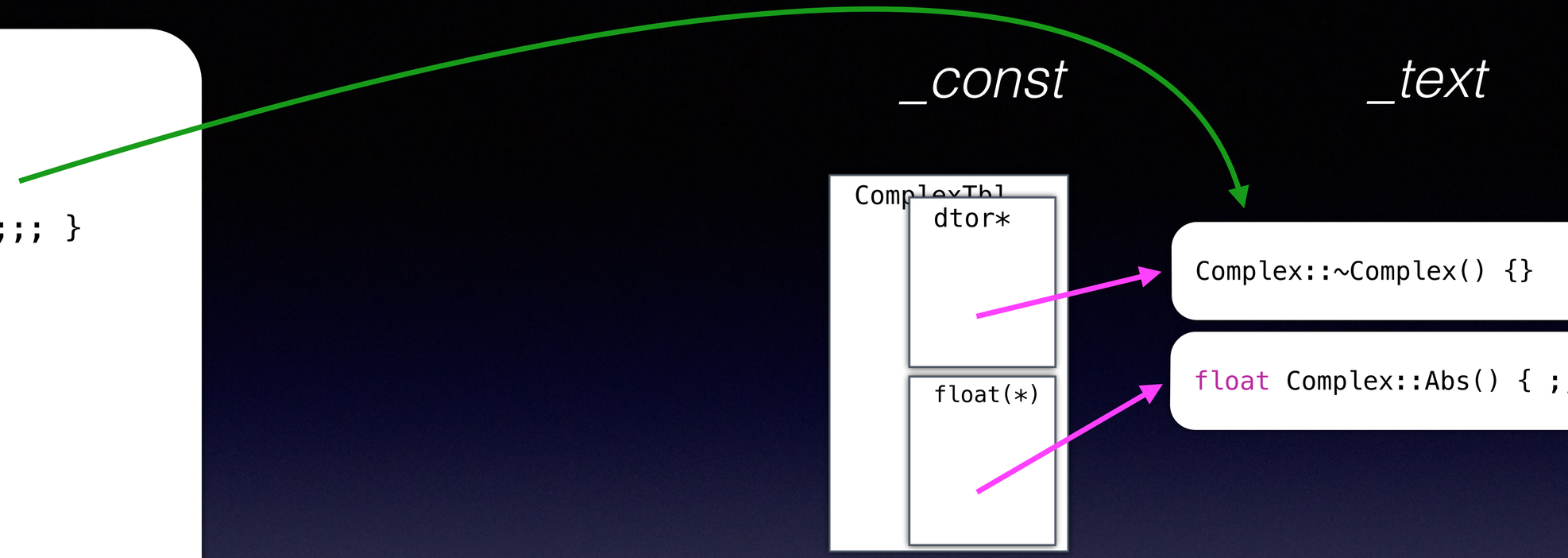
```
int main()
{
  Complex d;
}
```


```
Complex::~Complex() {}
```

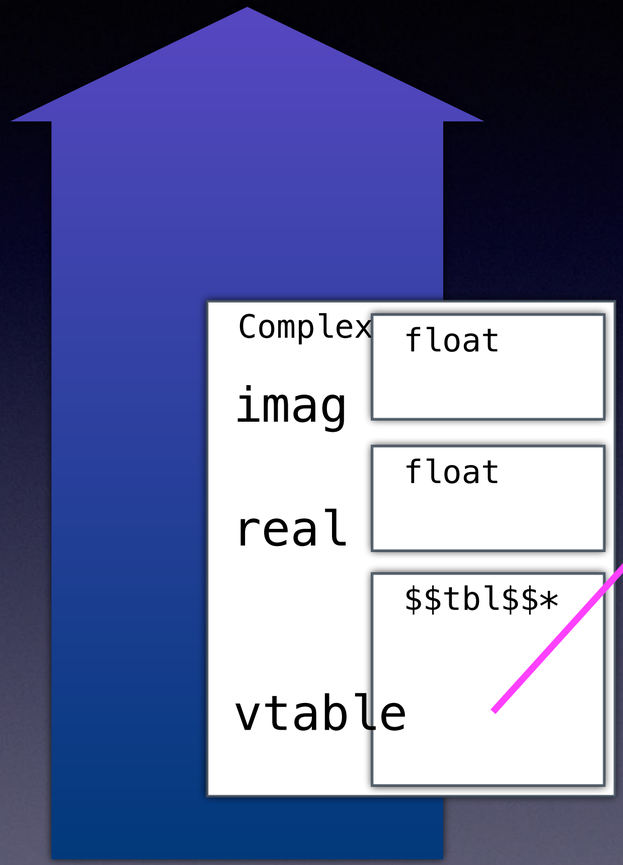
```
float Complex::Abs() { ;
```

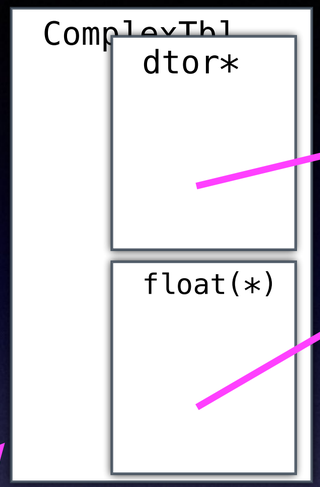
```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};
```

```
int main()
{
  Complex d;
}
```



```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};




int main()
{
  Complex d;
}
```





_const

_text

```
Complex::~Complex() {}
```

```
float Complex::Abs() { ;
```

YMMV

- From Working Draft N4431, Clause 9.2, Note 13:
- *Nonstatic data members of a (non-union) class with the same access control (Clause 11) are allocated so that later members have higher addresses within a class object. The order of allocation of non-static data members with different access control is unspecified (Clause 11). Implementation alignment requirements might cause two adjacent members not to be allocated immediately after each other; so might requirements for space for managing virtual functions (10.3) and virtual base classes (10.1).*
- Layout of Non-POD is implementation defined

ProTip!

When in doubt…

- Read the standard:
- http://open-std.org/JTC1/SC22/WG21/docs/papers/2015/n4431.pdf

```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() {}
  virtual float Abs() { ;;; }
  float angle;
};

int main()
{
  Complex c;
}
```

_const

_text

```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() {}
  virtual float Abs() { ;;; }
  float angle;
};

int main()
{
  Complex c;
}
```

_const

_text

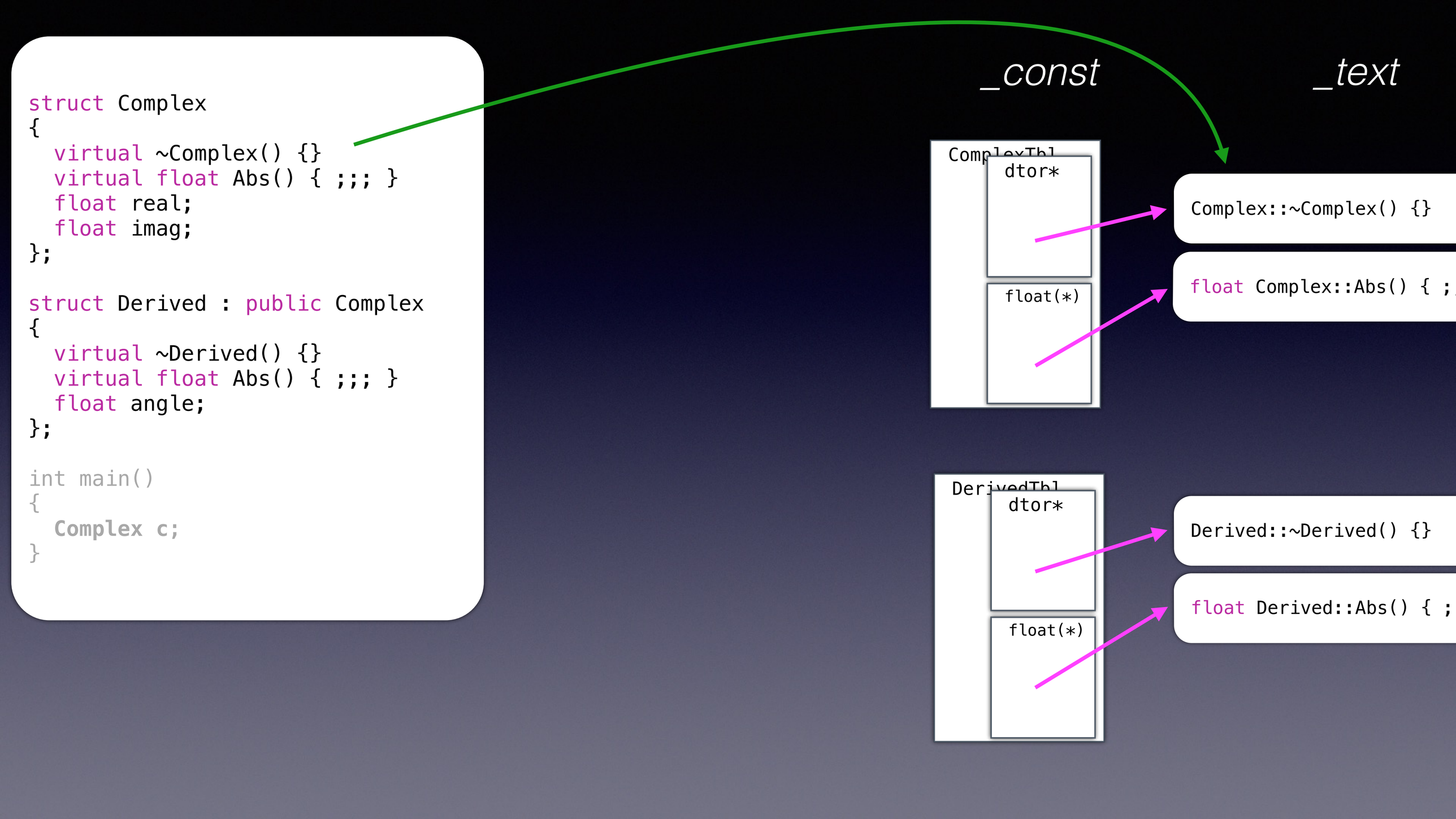
```
Complex::~Complex() {}
```

```
float Complex::Abs() { ;
```

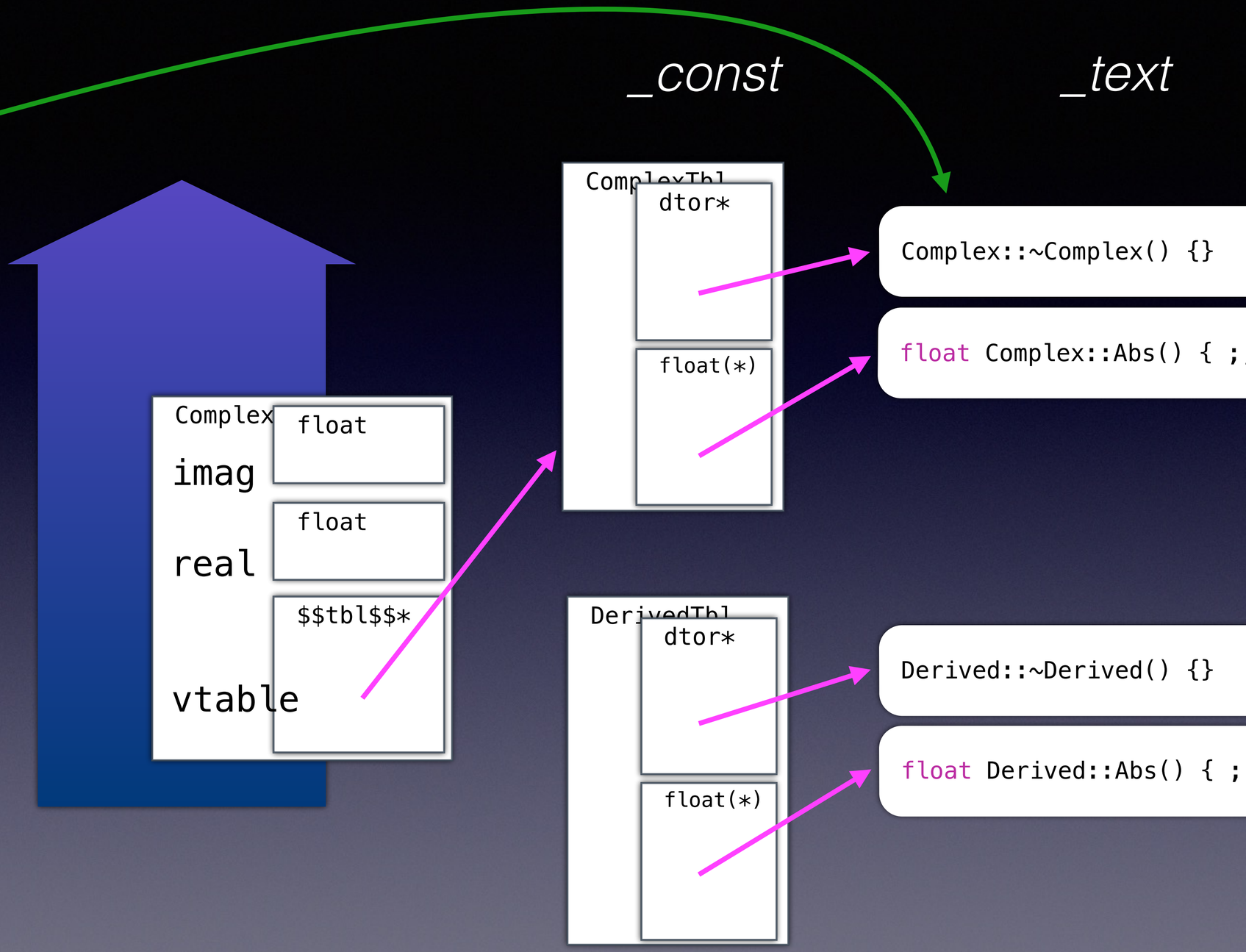
```
Derived::~Derived() {}
```

```
float Derived::Abs() { ;
```

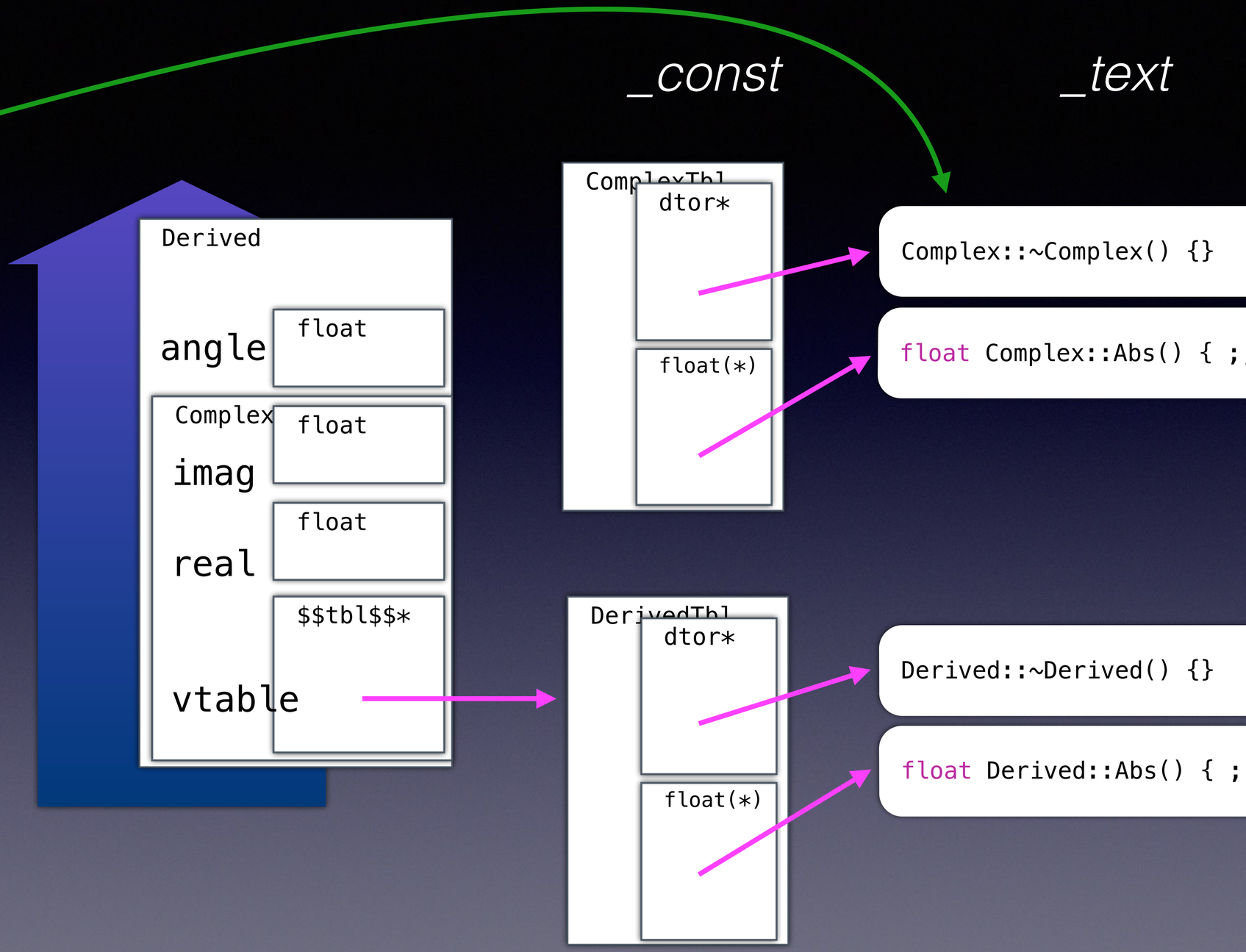
```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() {}
  virtual float Abs() { ;;; }
  float angle;
};

int main()
{
  Complex c;
}
```



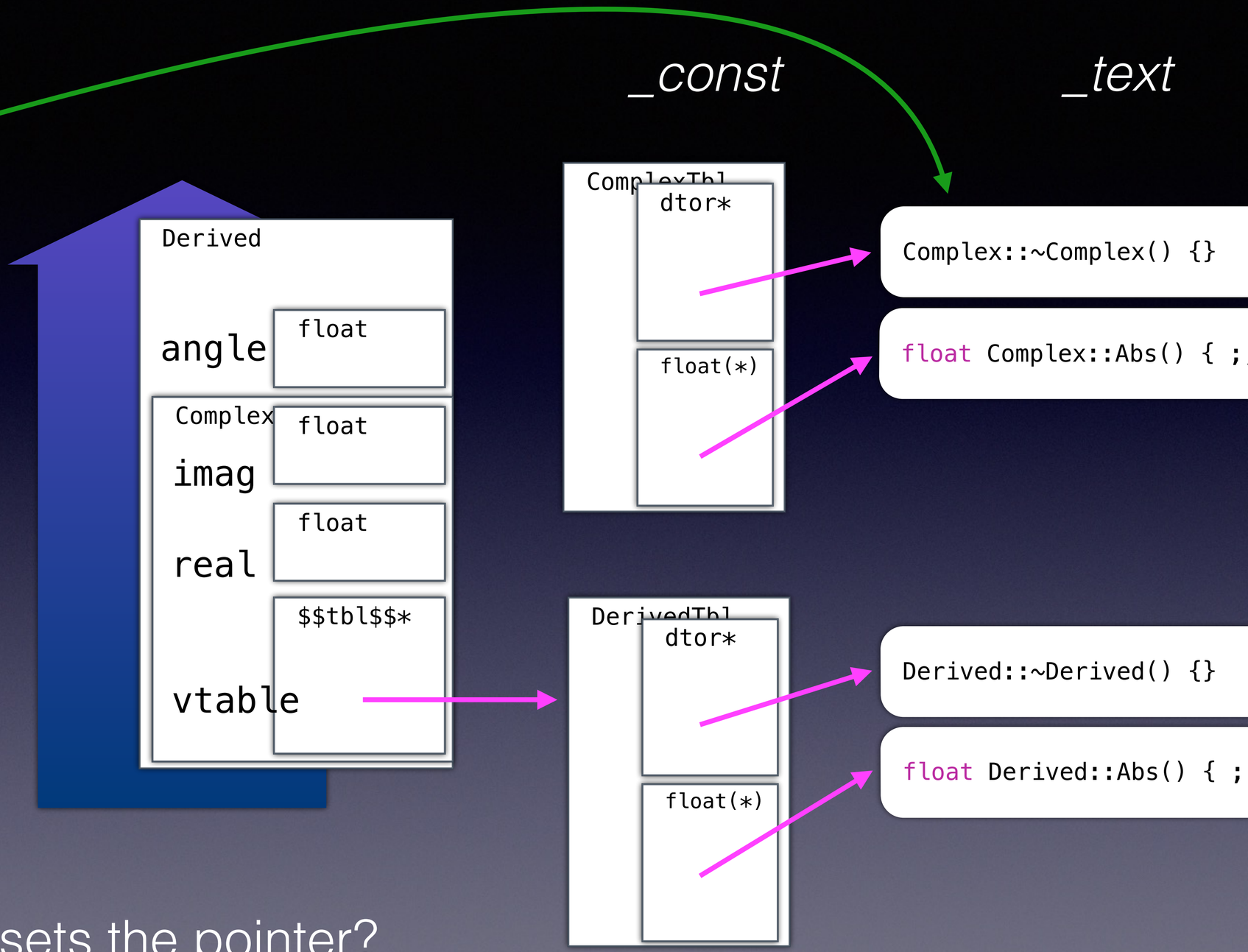
```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() {}
  virtual float Abs() { ;;; }
  float angle;
};

int main()
{
  Complex c;
}
```



```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() {}
  virtual float Abs() { ;;; }
  float angle;
};

int main()
{
  Derived c;
}
```



```
struct Complex
{
  virtual ~Complex() {}
  virtual float Abs() { ;;; }
  float real;
  float imag;
};

struct Derived : public Complex
{
  virtual ~Derived() {}
  virtual float Abs() { ;;; }
  float angle;
};

int main()
{
  Derived c;
}
```



Where is the code that sets the pointer?

C++ Constructors

- Virtual functions work by having pointers to functions (v-tables) initialized to the desired functions at run-time
- The job of the constructor

```
struct Complex {
  Complex();
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
Complex::Complex()
{
}
```

```
struct Complex {
  $$tbl$$ * vtable;
  Complex();
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
Complex::Complex()
{
}
```

```
struct Complex {
  $$tbl$$ * vtable;
  Complex();
  virtual ~Complex() = default;
  virtual float Abs();
  float real;
  float imag;
};
```

```
Complex::Complex()
{
  vtable = ComplexTbl;
}
```

```
struct Derived : public Complex {
  Derived();
  virtual ~Derived() = default;
  virtual float Abs();
  float angle;
};
```

```
Derived::Derived()
{
}
```

```
struct Derived : public Complex {
  Derived();
  virtual ~Derived() = default;
  virtual float Abs();
  float angle;
};
```

```
Derived::Derived()
{
}
```

```
struct Derived : public Complex {
  Derived();
  virtual ~Derived() = default;
  virtual float Abs();
  float angle;
};
```

```
Derived::Derived()
{
  this->Complex::Complex();
  vtable = DerivedTbl;
}
```

```
struct Derived : public Complex {
  Derived();
  virtual ~Derived() = default;
  virtual float Abs();
  float angle;
};
```

```
Derived::Derived()
{
  this->Complex
  vtable = Deri
}
```

```
Complex::Complex()
{
  vtable = ComplexTbl;
}
```

quiz_ctor.cpp:

```
#include <iostream>
using std::cout;

struct Erdos
{
  Erdos() { whoAmIReally(); }
  virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
  virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
  Erdos e;
  Fermat f;
}
```

```
$ make quiz_ctor
$ ./quiz_ctor
I really am Erdos
```

A

```
I really am Erdos
```

B

```
I really am Fermat
```

quiz_ctor.cpp:

```
#include <iostream>
using std::cout;

struct Erdos
{
  Erdos() { whoAmIReally(); }
  virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
  virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
  Erdos e;
  Fermat f;
}
```

```
$ make quiz_ctor
$ ./quiz_ctor
I really am Erdos
```

A

```
I really am Erdos
```

B

```
I really am Fermat
```

quiz_ctor.cpp:

```
#include <iostream>
using std::cout;

struct Erdos
{
  Erdos() { whoAmIReally(); }
  virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
  virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
  Erdos
  Fermat
}
```

```
Fermat::Fermat()
{
  this->Erdos::Erdos();
  vtable = FermatTbl;
}
```

```
$ make quiz_ctor
$ ./quiz_ctor
I really am Erdos
```

A

```
I really am Erdos
```

B

```
I really am Fermat
```

quiz_ctor.cpp:

```
#include <iostream>
using std::cout;

struct Erdos
{
  Erdos() { whoAmIReally(); }
  virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
  virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
  Erdos
  Fermat
}
```

```
Fermat
{
  this
  vtab
}
```

```
Erdos::Erdos()
{
  vtable = ErdosTbl;
  whoAmIReally();
}
```

```
$ make quiz_ctor
$ ./quiz_ctor
I really am Erdos
```

A

```
I really am Erdos
```

B

```
I really am Fermat
```

quiz_ctor.cpp:

```
#include <iostream>
using std::cout;

struct Erdos
{
  Erdos() { whoAmIReally(); }
  virtual void whoAmIReally() { cout<<"I really am Erdos\n"; }
};

struct Fermat : public Erdos
{
  virtual void whoAmIReally() { cout<<"I really am Fermat\n"; }
};

int main()
{
  Erdos
  Fermat
}
```

```
Fermat
{
  this
  vtab
}
```

```
Erdos::Erdos()
{
  vtable = ErdosTbl;
  whoAmIReally();
}
```

```
$ make quiz_ctor
$ ./quiz_ctor
I really am Erdos
```

A

```
I really am Erdos
```

B

```
I really am Fermat
```

NEVER call virtual functions in a constructor!!!

C++ Object model

- C++ object model is the way objects exist in memory
- Runtime polymorphism accomplished with vtables
  - Can dramatically increase the size of small objects
- Important to keep in mind, affects debugging and optimizing.
- Do not call virtual functions in an object’s constructor.

References

- http://en.wikipedia.org/wiki/Object-oriented_programming
- http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html
- Lippman, Stanley B., “Inside the C++ Object Model Paperback”, Addison-Wesley, 1996
- http://cppreference.com

Questions?