1. How can you quickly tell for a language with multiple inheritance, whether there is linearization applied?

2. How does the number of objects relate to the number of virtual tables?

3. When is the virtual table computed?

4. Is every class associated with a unique virtual table? What about virtual subtypes?

5. Given the following C++ classes:

   ```
   class E { public: virtual void f(E*); }
   class D { public: virtual void f(D*); }
   class C : virtual E { public: virtual void f(C*); }
   class B : D { public: virtual void f(B*); }
   class A : B, C { public: virtual void f(A*); }
   ```

   Assuming the existence of implementations of each instance of `f`, which of the following calls involves a virtual thunk?

   ```
   A a; E* e = a; e->f(e);  // A: E
   A a; D* d = a; d->f(d);  // A: D
   A a; C* c = a; c->f(c);  // A: E
   A a; B* b = a; b->f(b);  // A: B
   ```
Assignment 7.2 Linearization

Consider the following classes: A(B,C) B(D,E) C(F,G) D(G) E(F)

Give the linearization order of the following methods:

1. LPDFS with Duplicate Cancellation

Assignment 7.2 Linearization

Consider the following classes: A(B,C) B(D,E) C(F,G) D(G) E(F)

Give the linearization order of the following methods:

1. LPDFS with Duplicate Cancellation
2. Reverse Postorder Rightmost DFS
3. CI

Assignment 7.3 Multiple Inheritance – straight from the Exam 2015

Provide a C++ class structure and a main function, which fails to compile, due to multiple inheritance causing

1. ... an ambiguously resolvable call expression
2. ... ambiguous casting target types
Assignment 7.2 Linearization
Consider the following classes: \( A(B,C), B(D,E), C(F,G), D(G), E(F) \)
Give the linearization order of the following methods:
1. LPDFS with Duplicate Cancellation
2. Reverse Postorder Rightmost DFS
3. CI

Assignment 7.3 Multiple Inheritance - straight from the Exam 2015
Present a C++ class hierarchy and a class function which fails to compile, due to multiple inheritance causing:
1. ... ambiguous resolution of call expression
2. ... ambiguous casting type

Assignment 7.4 Multiple Inheritance - straight from the Exam 2014
This C++ code defines a few classes:

```cpp
class A {
  public:
    int a;
    virtual int f(int); // Virtual function declaration
    virtual int g(int); // Virtual function declaration
};

class B : public A {
  public:
    int b;
    int f(int); // Function declaration
    virtual int h(int); // Virtual function declaration
};

class C : virtual public A {
  public:
    int c;
    int f(int); // Function declaration
};

class D : public C, B {
  public:
    int d;
    int f(int); // Function declaration
};
```

This is the Virtual Table for class C:

```
Entry Value
0 | vbase_offset (40)
1 | offset_to_top (0)
2 | D RTTI
3 | int D::f(int)
4 | offset_to_top (-10)
5 | D RTTI
```

This is the Virtual Table for class D:

```
Entry Value
0 | vbase_offset (40)
1 | offset_to_top (0)
2 | D RTTI
3 | D B::f(int)
4 | offset_to_top (-40)
5 | D RTTI
6 | int D::f(int)
7 | int A::g(int)
8 | int D::h(int)
9 | vcall_offset (0)
10 | vcall_offset (-40)
11 | offset_to_top (-40)
12 | D RTTI
13 | int D::f(int)
14 | int A::g(int)
```

1. Draw the layout for a class object memory representation.
Assignment 7.5 Multiple Inheritance – straight from the Exam 2016

In this assignment, we program an interpreter for C++-Classes. The following C++-instruction sequence is our main concern; from that, we generate the corresponding Java-code to be interpreted with our framework:

```cpp
// C++ pc = new C();
Type C = Type::getTypedFor("C");
Pointer pc = Pointer::malloc(C::getSize());
C::getConstructor().callDirect(pc);
// &* pa = pc;
pa = f();
```

The signature of the Java-Classes/Interfaces used in this generated code can be found in the Appendix.

1. Consider the following C++-Classes:
```cpp
class A { public: int a; virtual void f(); }
class B : public A { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!
2. (IP) Given implementations of the methods castPointerTo and callVirtual for the type corresponding to C from directly above, that matches with the code generation and the representation/visible layout that you have determined above!

3. Consider the following C++ Classes:

```java
class A {
    public: int a; virtual void f();
}
class B {
    public: int b; virtual void f();
}
class C : public B, public A {
    public: int c; virtual void f();
}
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C.

4. Give implementations of the methods castPointerTo and callVirtual for the type corresponding to C from directly above, that matches with the code generation and the representation/visible layout that you have determined above!

5. One of the above virtual tables has a `break` as implementation for f. Which one?

Provide an implementation of the interface method Method::call, that performs the necessary actions in our framework, such that it is compatible with your representation/visible layout.
/** obtain the size in bytes for the type */

default int getSize() { ... }

public class Pointer {
  /** obtains fresh memory from the heap */
  public static Pointer malloc(int sizeInBytes) { ... }

  /** pointerArithmetic; add/sub returns the modified pointer + without changing this */
  public Pointer add(int offset) { ... }

  public Pointer sub(int offset) { ... }

  /** dereference the pointer and return whatever is found in the memory + you still need to cast to whatever is expected to be found there */
  public Object deref() { ... }
}

interface Method { // implementations are given by the framework
  Object callDirect(Pointer receiver, Object... parameters) { ... }
}

class A { public: int a; virtual void f(); }

class B { public: int b; virtual void f(); }

class C : public B, public A { public: int c; virtual void f(); }

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

4. Give implementations of the methods castPointerType and callVirtual for the type corresponding to C from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

5. One of the above virtual tables has a tweak as implementation for f. Which one? Provide an implementation of the interface Method::call that performs the necessary action in one framework, such that it is compatible with your representation/vtable layout.

1. Consider the following C++-Classes:

```cpp
class A { public: int a; virtual void f(); }

class B { public: int b; virtual void f(); }

class C : public B, public A { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

2. [IF] Give implementations of the methods castPointerType and callVirtual for the type corresponding to C from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

```cpp
C pc = new C();
C.getConcreteClassInfo();
A* pa = pc;
A* pc = C.castPointerType(pc, "A");
C.callVirtual(pa, "f");
```
The signatures of the Java-Classes/Interfaces used in this generated code can be found in the Appendix.

1. Consider the following C++-Classes:

```cpp
class A { public: int a; virtual void f(); }
class B : public A { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

2. [IF] Give implementations of the methods castPointerTo and callVirtual for the type corresponding to C from directly above, that matches with the code generation and the representation/table layout that you have determined above.

3. Consider the following C++-Classes:

```cpp
class A { public: int a; virtual void f(); }
class B { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

```cpp
C* pc = new C();
C* pn = pc;
A* pa = pc;
pa = f();
C::callVirtual((pa->f()));
```

The signatures of the Java-Classes/Interfaces used in this generated code can be found in the Appendix.

1. Consider the following C++-Classes:

```cpp
class A { public: int a; virtual void f(); }
class B { public: int b; virtual void f(); }
class C { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

2. [IF] Give implementations of the methods castPointerTo and callVirtual for the type corresponding to C from directly above, that matches with the code generation and the representation/table layout that you have determined above.