# Programming Languages

Dr. Michael Petter, Raphaela Palenta — WS 2018/19

**Exercise Sheet 8**

## Assignment 8.1 Quiz

1. Is every class associated with a unique virtual table? What about virtual subtables?

2. Given the following C++ classes:

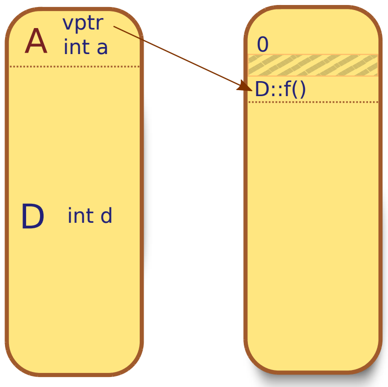
```
class E               { public: virtual void f(E*); };
class D               { public: virtual void f(E*); };
class C : virtual E { public: virtual void f(E*); };
class B : D           { public: virtual void f(E*); };
class A : B,C         { public: virtual void f(E*); };
```

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

   - ☐ `A a; E* e = &a; a.f(e);`
   - ☒ `A a; E* e = &a; e->f(&a);`
   - ☐ `A a; C* c = &a; a.f(c);`
   - ☐ `A a; C* c = &a; c->f(&a);`
   - ☐ `A a; B* b = &a; b->f(&a);`

3. Complete the object representation and virtual table diagrams given the following:

```
class A { public: int a; virtual void f(); }
class D : public A, public virtual V { public: int d; virtual void f(); }
class V { public: int v; virtual void f(); }
```



4. Consider the following code (note: shared base class):

```
class A { };
class B : public virtual A { };
class C : public virtual A { };
class D : public B, public C { };
...
C c; A* a = &c; (C*)a;
```

Is the cast correct?

### Suggested Solution 8.1

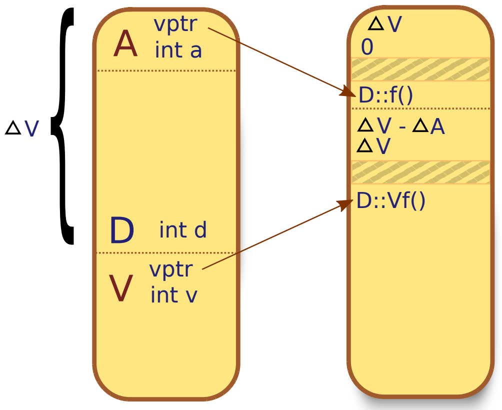
1. Yes, each class has exactly one corresponding virtual table. However, e.g., a class `A` may be a super-class of the classes `B` and `C` where the virtual subtable of `A` in the virtual tables of `B` and `C` is different. Consider the following code:

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

   The virtual subtable of `A` in the virtual table of `B` contains an entry for function `f` which delegates to the function `f` in class `B`. Whereas the virtual subtable of `A` in the virtual table of `C` contains an entry for function `f` which delegates to the function `f` in class `C`.

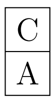
2. see before...

3.

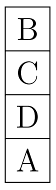


4. No, the cast is not correct. At compile time we do not know the exact distance from the class `A` to `C`. But a dynamic cast works!

Memory layout of an object C:



Memory layout of an object D:



### Assignment 8.2 Multiple Inheritance

This C++ code defines a few classes:

```cpp
class A {
public:
  int a;
  virtual int f(int);
  virtual int g(int);
};
class B : public A {
public:
  int b;
  int f(int);
  virtual int h(int);
};
class C : virtual public A {
public:
  int c;
  int f(int);
};
class D: public C, B {
public:
  int d;
  int f(int);
};
```

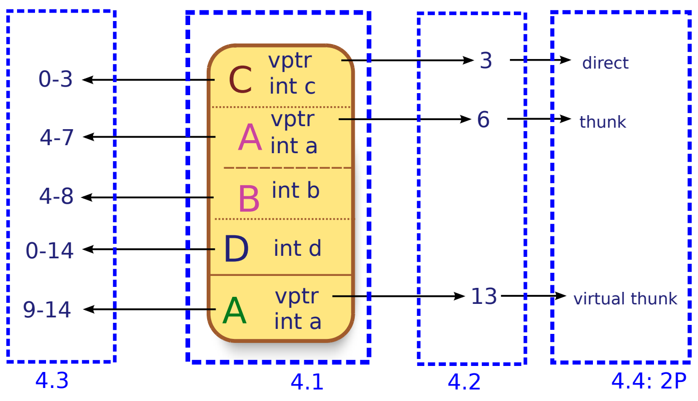
This is the Virtual Table for class `D`:

```
Entry| Value
   0 | vbase_offset (40)
   1 | offset_to_top (0)
   2 | D RTTI
   3 | int D::f(int)
   4 | offset_to_top (-16)
   5 | D RTTI
   6 | int D::f(int)
   7 | int A::g(int)
   8 | int B::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 `D` object memory representation!
2. For each `vptr`-attribute in your drawing, give the *entry number* to which `Vtable`-entry this pointer is pointing.
3. `D`'s virtual table is composed of several subtables. Your object memory representation is also composed of several parts, corresponding to particular subclasses.

   For each subclass part of your memory represenation, give the *entry numbers*, where the corresponding subtable within `D`'s virtual table *starts* and *ends*.
4. Thunks were not highlighted in the virtual table. Compare the entries 3, 6 and 13 in the virtual table. Which of them are thunks, which are virtual thunks, and which are direct addresses of `D::f(int)`?

### Suggested Solution 8.2



0-3
4-7
4-8
0-14
9-14
C
vptr
int c
A
vptr
int a
B
int b
D
int d
A
vptr
int a
3
6
13
direct
thunk
virtual thunk
4.3
4.1
4.2
4.4: 2P

**Assignment 8.3 Multiple Inheritance I**

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

1. ... an ambiguously resolvable call expression
2. ... ambiguous casting target types

**Suggested Solution 8.3**

```
class D              {                    };
class C              { public: void f(); };
class B : virtual public D { public: void f(); };
class A : public B, public C      {                    };
int main(){
  A *a = new A();
  a->f(); // error: request for member 'f' is ambiguous

  D* d = &a;
  a = (A*)d; // error: cannot convert from pointer to base class 'D' to
          // pointer to derived class 'A' because the base is virtual
  return 0;
}
```

**Assignment 8.4 Multiple Inheritance II**

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:

```
C* pc = new C();
A* pa = pc;
pa->f();
```

⇒

```
// C* pc = new C();
Type C = Type.getTypeFor("C");
Pointer pc = Pointer.malloc(C.getSize());
C.getConstructor().callDirect(pc);
// A* pa = pc;
Pointer pa = C.castPointerTo(pc,"A");
// 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:

   ```
   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. 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/vtable layout that you have determined above!

3. Consider the following C++-Classes:

```
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(); }
```

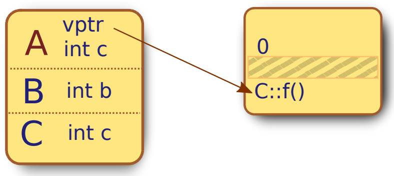
   Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

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/vtable layout that you have determined above!

5. One of the above virtual tables has a *thunk* as implementation for `f`. Which one? Provide an implementation of the interface method `Method::callDirect`, that performs the necessary actions in our framework, such that it is compatible with your representation/vtable layout.

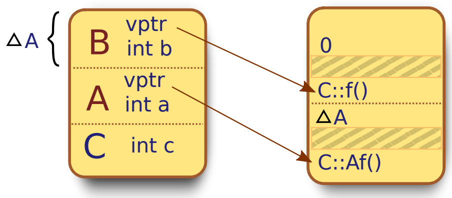
## Suggested Solution 8.4

1.



2.
```
Object callVirtual(Pointer p, String m, Object... ps){
  Pointer vtable = (Pointer)p.deref();
  Method f = (Method)vtable.deref();
  return f.callDirect(pc,ps);
}
Object castPointerTo(Pointer p, String c){
  return p;
}
```

3.



4.
```
Object callVirtual(Pointer p, String m, Object... ps){
  Pointer vtable = (Pointer)p.deref();
  Method f = (Method)vtable.deref();
  return f.callDirect(pc,ps);
}
Object castPointerTo(Pointer p, String c){
  if ("A".equals(c)) return p.add(Type.getTypeFor("B").getSize());
  return p;
}
```

5.
```
Object callDirect(Pointer receiver, Object... parameters){
  Pointer pc = receiver.sub(Type.getTypeFor("B").getSize());
  return Type.getTypeFor("C").callVirtual(pc,"f",parameters);
}
```

**Appendix for Assignment 4**:

Let the following Java-Interfaces be given as an API for the runtime components of our interpreter:

```java
interface Type {
  /** obtain an object, representing the type denoted by the name t */
  default Type getTypeFor(String t) { ... }
  /** perform a virtual method call to method m on p with params ps */
  Object callVirtual(Pointer p,String m, Object... ps);
  /** returns a pointer to the cast target c wrt. the current type/pointer */
  Pointer castPointerTo(Pointer p,String c);
  /** obtain the constructor for the type, given there is one */
  Method getConstructor();
  /** 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) { ... }
  /** pointerarithmetics; add/sub returns the modified pointer
   *  without changing this
   */
  public Pointer add(int offset) { ... }
  public Pointer sub(int offset) { ... }
  /** derefenciate 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) { ... }
}
```