Multiple and Virtual Inheritance in C++

- Initialization of virtual bases
- Frozen classes in C++
- Construction order with multiple inheritance
Constructors and virtual base classes?

```cpp
struct V {  
    V(const char * s) {  
        cout << s;  
    }  
};

struct B1 : virtual V {  
    B1(const char * s) : V("B1") {  
        cout << s;  
    }  
};

struct B2 : virtual V {  
    B2(const char * s) : V("B2") {  
        cout << s;  
    }  
};

struct D : B1, B2 {  
    D() : B1("DB1"), B2("DB2") {}  
} d;
```

What will be printed?
Construction of a virtual base class

- **Answer:** nothing will be printed. The compiler will issue an error message.
- **Work-around 1:** define a default constructor in V.
- **Work-around 2:** call V’s constructor directly from the constructor of D.
  - Virtual bases are always initialized by most derived class – other initializations are ignored. This also applies if the most derived class is not an immediate base class of the virtual base.
  - Work-around 1 is actually the same solution – by giving V a default constructor, it will always be implicitly called by the most derived class.
- **Comments:**
  - All virtual inheritances of the same object are unified.
  - All non-virtual inheritances of the same object are distinct.
**Virtual base initialization example**

```cpp
struct V {
    V();
    V(int);
    ...};
struct A : virtual V {
    A();
    A(int i): V(i) { /* ... */ }  
    ...};
struct B : virtual V {
    B();
    B(int i) { /* ... */ }  
    ...};
struct C : A, B {
    C(int i): V(i) { /* ... */ }
    ...};
V v(1); // use V(int)
A a(2); // use V(int)
B b(3); // use V()
C c(4); // use V(int)
```
Frozen classes

```c
#ifndef ICE
struct ice__ {
    ice__() {}; // Constructor
};
#define ICE private virtual ice__
#endif ICE

#include "ice.h"

class Frozen: ICE {
    // ...
};
class Violation: public Frozen {
    // ...
};

Error: ice__::ice__() is not accessible in function Violation::Violation()
```

The trick may be easily worked around by virtually deriving the Violation class from ice__
Construction and multiple inheritance

- **Elements to initialize:** sub-objects and fields.
- **Where to initialize?** Best is in the initialization list (after the constructor signature). Sub-objects can only be initialized in the initialization list.
  - Order of elements in the initialization list is unrelated to the order in which they will actually be invoked. This makes it possible to guarantee that the construction order is the exact opposite of destruction order.
- **Construction order** is a recursive algorithm:
  1. Virtual base classes, in the order they occur in depth-first, left-to-right (by definition order) traversal of the hierarchy graph.
     - If a virtual base class is derived from a non-virtual base, then this non-virtual base will be constructed before the virtual base.
  2. Remaining base classes, in the order they occur in the hierarchy graph.
  3. Fields (data members).
  4. Constructor body.
- **Order of destruction** is the same in reverse.
Initialization order algorithm example

- Apply topological sort ranking inheritance DAG in a depth-first, left-right scan
  - Virtual and non-virtual inheritance are treated alike.
- Construct all virtual base classes (immediate and non-immediate)
  - Use ranking order.
  - Do not construct twice.
  - Apply recursively to construct their non-virtual bases.
- Construct non-virtual immediate base classes:
  - Use ranking order (same as definition order).
  - Apply recursively to construct their non-virtual bases.
- Construction order in example:
  - \text{U1 U2 Y X V2 V1 V3 V4 B1 B2 D}
- Destruction order in example:
  - \text{D B2 B1 V4 V3 V1 V2 X Y U2 U1}