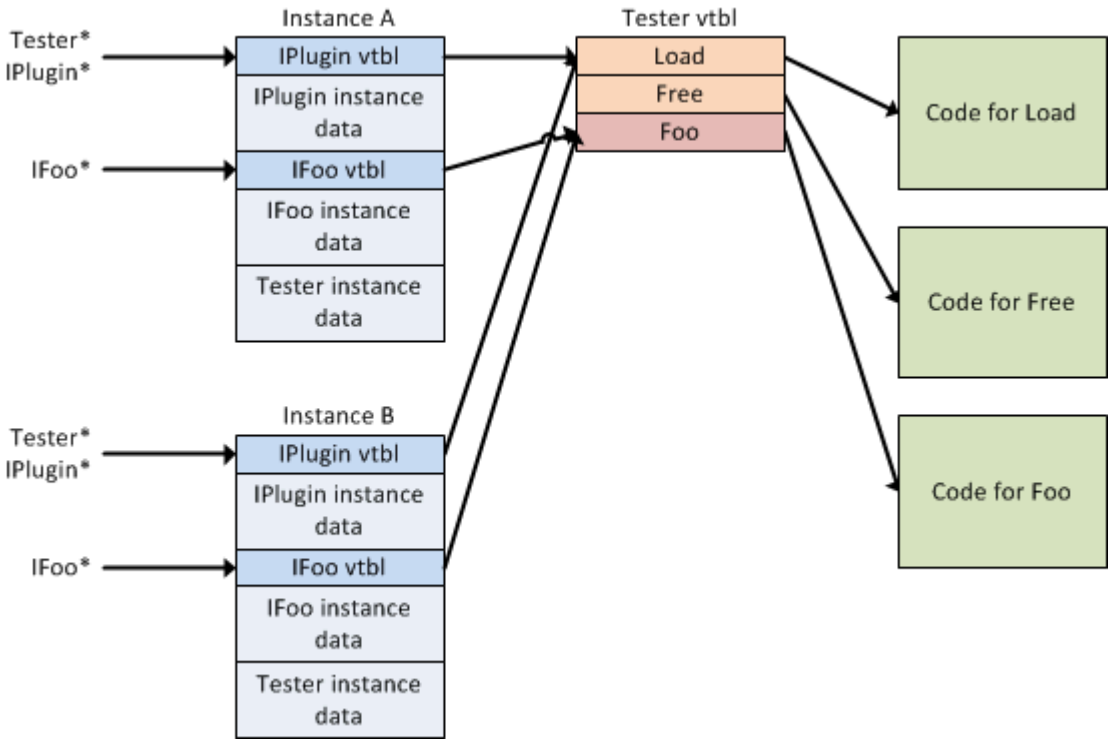
Report on vTable

1/ Definition:

Virtual Table is a lookup table of function pointers used to dynamically bind the virtual functions to objects at runtime. It is not intended to be used directly by the program, and as such there is no standardized way to access it.

Virtual tables in C++ is an implementation detail. One possible implementation is shown on the diagram below.



Two instances of the class (A and B) exists. Each instance has two virtual table pointers and the virtual table contains pointers to actual code.

2/ Virtual Table:

- Every class that uses virtual functions (or is derived from a class that uses virtual functions) is given it's own virtual table as a secret data member.

- This table is set up by the compiler at compile time.

- A virtual table contains one entry as a function pointer for each virtual function that can be called by objects of the class.

- Virtual table stores NULL pointer to pure virtual functions in ABC.

- Virtual Table is created even for classes that have virtual base classes. In this case, the vtable has pointer to the shared instance of the base class along with the pointers to the classe's virtual functions if any.

3/ Virtual pointer(_vptr):

- This vtable pointer or _vptr, is a hidden pointer added by the Compiler to the base class. And this pointer is pointing to the virtual table of that particular class.

- This _vptr is inherited to all the derived classes.

- Each object of a class with virtual functions transparently stores this _vptr.

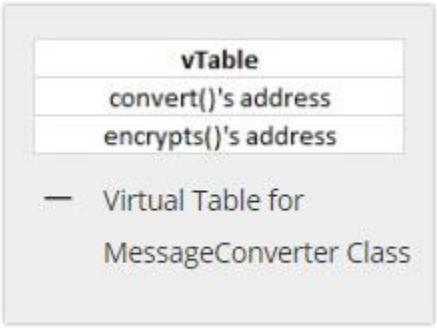
- Call to a virtual function by an object is resolved by following this hidden _vptr.

4/ How virtual functions works internally using vTable and vPointer?

Lets look at an example,

```
class MessageCoverter
{
        int m_count;
public:
        virtual std::string convert(std::string msg)
        {
                msg = "[START]" + msg + "[END]";
                return msg;
        }
        virtual std::string encrypt(std::string msg)
        {
                return msg;
        }
        void displayAboutInfo()
        {
                std::cout<<"MessageCoverter Class"<<std::endl;
        }
};
```

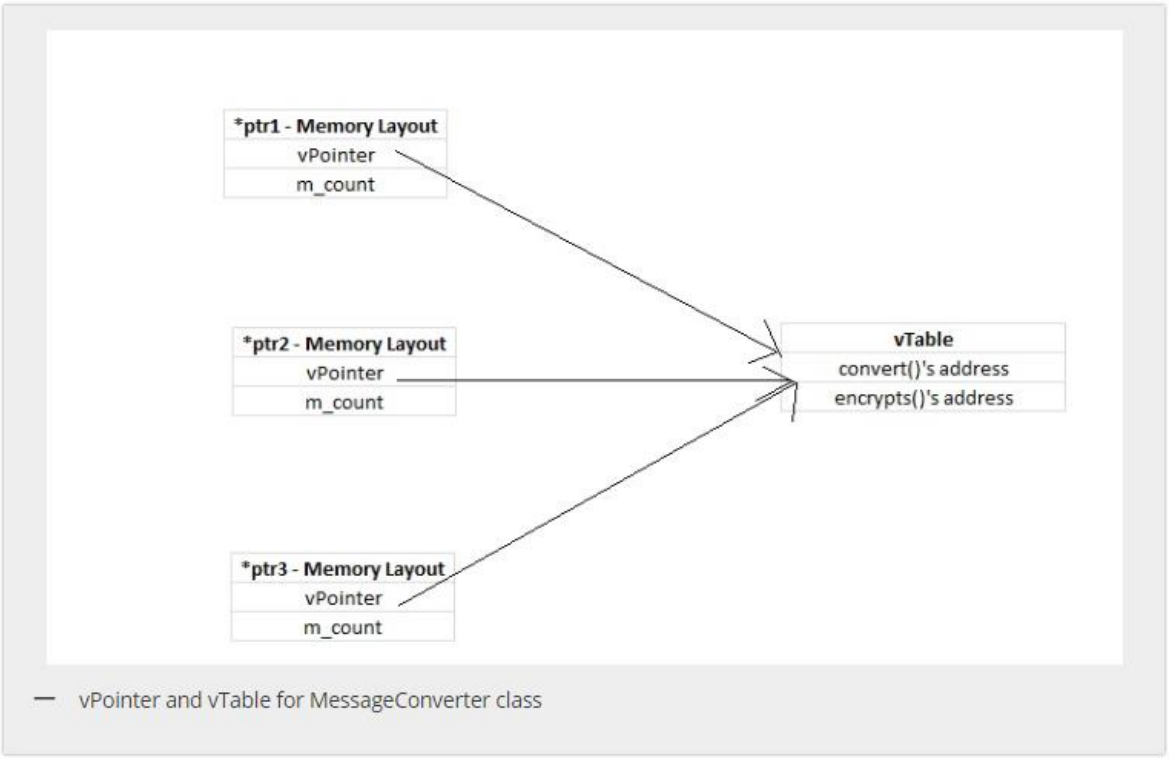
As the above class contains more than 0 virtual functions i.e. convert() and encrypt(), hence it has a vTable associated with it. vTable will look like this,



Now suppose we created three different objects for this class,

```
1 MessageCoverter * ptr1 = new MessageCoverter();
2 MessageCoverter * ptr2 = new MessageCoverter();
3 MessageCoverter * ptr3 = new MessageCoverter();
```

Memory layout of each of the object is as follows,



— vPointer and vTable for MessageConverter class

Because for virtual functions linking was not done at compile time. So, what happens when a call to virtual function is executed ,i.e.

Steps are as follows,

1/ vpointer hidden in first 4 bytes of the object will be fetched

2/ vTable of this class is accessed through the fetched vPointer

3/ Now from the vTable corresponding function's address will be fetched

4/ Function will be executed from that function pointer

5/ References

https://stackoverflow.com/questions/5868431/what-is-the-structure-of-virtual-tables-in-c

https://www.go4expert.com/articles/virtual-table-vptr-t16544/

https://thispointer.com/how-virtual-functions-works-internally-using-vtable-and-vpointer/

https://www.quora.com/What-are-vTable-and-VPTR-in-C++