Vptr and Vtable

To implement virtual functions, C++ uses a special form of late binding known as the virtual table or vTable. The virtual table is a lookup table of functions used to resolve function calls in a dynamic/late binding manner.

Every class that uses virtual functions (or is derived from a class that uses virtual functions) is given its own virtual table.

This table is simply a static array that the compiler creates at compile time. A virtual table contains one entry for each virtual function that can be called by objects of the class.

Each entry in this vTable is simply a Function Pointer that points to the most-derived function accessible by that class ie the most Base Class.

The compiler also adds a hidden pointer to the base class, which we will call *__vPtr.

*__vPtr is set (automatically) when a class instance is created so that it points to the virtual table for that class. *__vPtr is inherited by derived classes,

Exception Handling

One of the advantages of C++ over C is Exception Handling. Exceptions are run-time anomalies or abnormal conditions that a program encounters during its execution. There are two types of exceptions: a)Synchronous, b)Asynchronous(Ex:which are beyond the program's control, Disc failure etc).

try: represents a block of code that can throw an exception.

catch: represents a block of code that is executed when a particular exception is thrown.

throw: Used to throw an exception. Also used to list the exceptions that a function throws, but doesn't handle itself.

Const Keyword

Constant Variables:
There are a certain set of rules for the declaration and initialization of the constant variables:

- The const variable cannot be left un-initialized at the time of the assignment.
- It cannot be assigned value anywhere in the program.
- Explicit value needed to be provided to the constant variable at the time of declaration of the constant variable.

Final Keyword

we can use final for a function to make sure that it cannot be overridden.

We can also use final in C++ to make sure that a class cannot be inherited.

```
class Base
{
public:
    virtual void myfun() final
    {
```

```cpp
        cout << "myfun() in Base";
    }
};
class Derived : public Base
{
    void myfun()
    {
        cout << "myfun() in Derived\n";
    }
};

int main()
{
    Derived d;
    Base &b = d;
    b.myfun();
    return 0;
}
```

```cpp
class Base final
{
};

class Derived : public Base
{
};

int main()
{
    Derived d;
    return 0;
}
```

Shallow Copy and Deep copy

In general, creating a copy of an object means to create an exact replica of the object having the same literal value, data type, and resources.

- Copy Constructor
- Default assignment operator

In shallow copy, an object is created by simply copying the data of all variables of the original object. This works well if none of the variables of the object are defined in the heap section of memory. If some variables are dynamically allocated memory from heap section, then copied object variable will also reference then same memory location.

In Deep copy, an object is created by copying data of all variables and it also allocates similar memory resources with the same value to the object. In order to perform Deep copy, we need to explicitly define the copy constructor and assign dynamic memory as well if required. Also, it is required to dynamically allocate memory to the variables in the other constructors, as well.