

Virtual Functions Part Three Solutions

- Briefly describe how virtual member functions are implemented
 - When the compiler encounters a class with a virtual function, it generates some code which will check the dynamic type at runtime
 - It also creates an array known as a “vtable”
 - The elements of this array are pointers to the virtual member functions of the class
 - At runtime, the program loads the vtable for the dynamic type, looks up the appropriate virtual member function and calls it

- Give one advantage of using virtual member functions
 - Virtual member functions allow us to call member functions for an instance's dynamic type
 - e.g. container of pointers to base class instances
- Give one disadvantage of using virtual member functions
 - Overhead
 - Virtual member function calls are slower than non-virtual
 - Classes with virtual member functions use more memory

- Explain why a base class should always have a virtual destructor implemented
 - The default destructor generated by the compiler is non-virtual
 - This means that static binding is always used for the destructor
 - If called on a pointer or reference which has a different dynamic type, this means that the derived class part of the instance is not destroyed, resulting in a partially destroyed object
 - Providing a virtual destructor will ensure that dynamic binding is always used

- In the program you wrote in the previous lesson, add a non-virtual destructor function to your classes, including the child classes. Something like this:

```
class Drawable {  
public:  
    virtual void draw() { cout << "I'm a Drawable shape!\n"; }  
    ~Drawable() { cout << "Drawable shape says goodbye!\n"; }  
};
```

- What output do you expect to see?
- Compile and run the program and compare the results

- Compile and run the program and compare the results
 - Output is
 - I'm a Circle!
 - I'm a Triangle!
 - Drawable shape says goodbye!
 - Drawable shape says goodbye!
 - Note that the Circle and Triangle destructors are not called

- Now make the destructor of Drawable virtual:

```
class Drawable {  
public:  
    virtual void draw() { cout << "I'm a Drawable shape!\n"; }  
    virtual ~Drawable() { cout << "Drawable shape says goodbye!\n"; }  
};
```

- What output do you expect to see?
- Compile and run the program and compare the results
- Explain any differences from the previous output
- Why is this important?

- **Actual output**

I'm a Circle!

I'm a Triangle!

Circle says goodbye!

Drawable shape says goodbye!

Triangle says goodbye!

Drawable shape says goodbye!

- When the base class destructor is declared virtual, the derived class destructors are called as well
- This ensures that all the objects are properly destroyed
- There are no memory leaks or other undefined behaviour