```
1  ///////// Lecture 70: Initialisation vs Assignment
2  // This will help you decide which one to use and in
   previous videos we've already seen a little bit
   about initialisation and assignment.
3  // In the below example:
4  int main(){
5      Integer a = 5; // Initialisation
6
7      a = 6; // Assignment
8  }
9  // What if we prefer initialisation? We are
   initialising the object a with some value: so
10 int main(){
11     Integer a1 = 5; // Initialisation
12     Integer a2 {6}; // Initialisation
13 }
14 // It invokes the constructor at the point of the code
   at which it is invoked, and it is destroyed at the
   end of the scope. And that's why we see the
   destructor call.
15
16 // Compare the above to this:
17 int main(){
18     Integer a1; // create an instance first.
19
20     a1 = 5; // Assign it a value
21 }
22 // When you build this and run, you'll see the
   following output:
23 // A constructor call.
24 // Then you also see a parameterised constructor call:
25 // the parameterised constructor call is for the
   temporary object that gets created on the right hand
   side of the assignment: moore specifically:
26 a1 = Integer(5);
27 // Since Integer(5) is a temporary, the next call is
   the move assignment operator.
28 // After the assignment is done, then you see the
   destructor.
29
30 // So obviously assignment requires more function
   calls initialisation. This is why you should always
```

```
    prefer initialisation over assignment.

// Consider a class called product

class Product{
    Integer m_Id;
public:
    Product (const Integer &id){
        stD::cout << "Product(const Integer &)" <<
          std::endl;
        m_Id = id;
    }
    // Create a destructor
    ~Product(){
        std::cout << "~Product()" << std::endl;
    }
};
int main(){
    Product p(5);
}
/* Let's refer to the output:
1. Integer(int) --> parameterised call of the Integer
  object because the 5 gets converted into an Integer
  object. Done automatically by the compiler.
2. Default call to the constructor of Integer - that
  is because we are default constructing the m_Id
  Integer.
3. Product(const Integer &) --> constructor for product
4. Copy Assignment constructor -->  We are using
  assignment here to initialise the Integer object.
*/


// How do we use initialisation instead of assignment?
  That would be possible by initialising the Integer
  object in a list called as member initialiser list.
  It is a list that occurs immediately after the
  constructor.
class Product{
    Integer m_Id;
public:
    Product(const Integer &id): m_Id(id){
```

```
62            // the assignment here is no longer required.
63        }; // When the compiler sees this, it will first
          • construct the m_Id Integer object and initialise
          • it with this value that you have passed into the
          • constructor, and then it'll create the enclosing
          • Product object.
64        // Always use the member initialiser list to
          • initialise the members of your class. You can
          • use the initialiser list for initialising any
          • kind of type (primitive types too), and you can
          • initialise it in any order. they are initialised
          • in the order in which they are declared.
65    };
66
```