```cpp
// Lecture 51: Default and Deleted Functions
#include <iostream>

// Integer class
class Integer {
    int m_Value {}; // create this variable, initialise its value to 0.
    // Recall that in C++11, if you declare any user defined constructors, the default constructor won't be created.
public:
    // Integer() {
    //  m_Value = 0;
    // }
    // If you comment out the constructor above, and try to build the code it will fail at the first line of main()- because a no argument constructor doesn't exist.

    // However, in C++11, you can request the compiler to generate a default implementation of some functions, so we can do that by writing the declaration of the constructor and using the keyword 'default', so I'm assigning default to the declaration of the integer constructor and the compiler will create a default implementation of this constructor without having to define it manually.
    Integer() = default;
    // Now our code builds fine.
    // You can use the default keyword with only those functions that can be synthesised by the compiler, so that includes the destructor, copy constructor, and assignment operator.

    // It can be seen in assembly that the compiler has automatically created a definition for this integer.
    Integer(int value) {
        m_Value = value;
    }
    // in the same way we can ask the compiler to
```

```
        create a default implementation of the copy
        constructor:
    Integer(const Integer &) = default;
    //In our case, the compiler does it implictly for
        us. But in some cases this is required - so
        we'll see those cases in subsequent sessions.
    // The other important keyword is in fact the
        'delete' keyword: in the context of the classes.

    // Imagine that we don't want to create the copy
        of the integer object. If we try to create a
        copy of i1 for example, this should NOT be
        allowed. So even if we remove this, the compiler
        is anyway going to synthesise the above code
        line instead.

    // To tell the compiler not to synthesise the copy
        constructor, we use the word 'delete'. This
        tells the compiler to not to synthesise the copy
        ctor.
    Integer(const Integer &) = delete;
    // If we try to build this code now, it gives an
        error: Attempting to reference a deleted
        function...
    //If we want to prevent copying of this object
        then we should also mark the assignment operator
        as deleted.
    // Unike default, delete can be used on any kind
        of function.
    void SetValue(int value){
        m_value = value;
    }
    // To prevent the callers from passing float
        values to SetValue(), we can declare a function
        SetValue() with float type as an argument and
        assign 'delete' to it.
    void SetValue(float) = delete;
};

int main() {
    Integer i1; // default constructor invoked
    Integer i2(5); // parameterised constructor invoked
```

```
45          i1.SetValue(5);
46          // Some users can also pass a float value to the
            •  function without compiler errors (may have
            •  compiler warning)
47          i1.SetValue(2.5f);
48          // To prevent the callers from passing float
            •  values to SetValue(), we can declare a function
            •  SetValue() with float type as an argument and
            •  assign 'delete' to it.
49          // After doing this, the code will not compile,
            •  because the compiler will now match this call to
            •  a function SetValue() that accepts a float but
            •  since that has been marked as delete, it cannot
            •  be invoked.
50
51          return 0;
52      }
53
```