```
1  ///////// Lecture 68: Type Conversion II - Basic and
   •   User Defined Types
2  #include "Integer.h" // our beloved integer.
3
4  int main(){
5      Integer a1{5}; // create a user defined a1, and
   •     initialise a1 with some primitive value. This
   •     expression invokes the parameterised constructor
   •     of the integer class and initialises this object
   •     with the value 5.
6
7      // In this way, the primitive type gets converted
   •     from one type into another, so constructors also
   •     take part in type conversion.
8      // So constructors also take part in type
   •     conversion, and they can be invoked explicitly
   •     or implicitly.
9
10     // In the above case, we invoke the parameterised
   •     constructor of the integer class explicitly.
11     // it may get invoked implicitly if we write an
   •     expression like this:
12
13     Integer a2 = 5; // This does not directly invoke
   •     the parameterised constructor of the integer. To
   •     initialise the a2 object with this value, the
   •     compiler will implicitly search for a
   •     parameterised constructor in the Integer class
   •     and when it finds one, it will invoke it.
14
15     Integer a3 = "abc"; // If we try to assign a
   •     string to this Integer, the compiler will search
   •     for a constructor in the Integer class that
   •     accepts a string type. Because we don't have
   •     such a constructor, the code cannot compile.
16     // ERRORS: - INITIALISING: cannot convert from
   •     'const char[4]' to 'Integer'.  - No suitable
   •     constructor exists to convert from "const char
   •     [4]" to "Integer".
17     // Therefore, whenever you use the assignment
   •     operator to initialise an object like this, the
   •     compiler will implcitly invoke the corresponding
```

```
        parameterised constructor.
}

//There are other cases also where the compiler may
  automatically invoke the parameterised constructor.
void Print(Integer a){...};
int main(){
    Print (5); // Even if we invoke this function with
      a primitive type, this also works! So compiler
      will convert this primitive type into this user
      defined type by invoking its parameterised
      constructor.
    return 0;
}
// And the same thing would have happened if we had
  been accepting this object as a constant reference.
void Print(const Integer &a){...};
int main(){
    Print (5); // This works too!
    return 0;
}

// The other case is if we try to assign a primitive
  type to a user defined type through the assignment
  operator.
int main(){
    Integer a1{5};
    a1 = 7;
    // Even though we do not have an assignment
      operator that accepts a primitive type, but we
      have two assignment operator overloads - one
      accepts a constant integer reference, and the
      other accepts a R-value reference of an Integer
      object.

    // Since 7 is an R-value the call a1 = 7 will
      match the move assignment.

    // So this object will be constructed through its
      parameterised constructor and then the temporary
      will be moved into the object on the left hand
      side.
```

```
42  }
43
44  // Notice the difference between the following two
      statements:
45  int main(){
46      Integer a2 = 5; // Initialisation
47      a2 = 7;  // Assignment
48  }
49  // We will revisit the concepts of initialisation vs
      assignment in the subsequent lectures. Because the
      compiler automatically uses the parameterised
      constructor to convert the primitive type into user
      defined type, in some cases that may be undesirable,
      we do not want the coompiler to automatically use
      our constructor for implicit conversion. Therefore,
      we can MARK the constructor with the explicit
      keyword in Integer.h.
50
51  // In integer.h
52  class Integer{
53      ...
54  public:
55      ...
56      explicit Integer(int value);
57      // Therefore we can mark the constructor with the
          explicit keyword.
58  };
59  // The compiler can no longer use that particular
      constructor for implicit type conversion.
60
61  int main(){
62      Integer a2 = 5; // gives you an error!!
63      a1 = 7; // and you can see both of these lines
          showing some kind of error, and the error is
          that NO SUITABLE CONSTRUCTOR EXISTS to convert
          'int' to 'Integer'.
64      // In most cases, the single argument constructors
          of your class should be marked with the explicit
          keyword, but not all classes may do that because
          in some cases, we have classes that are thin
          wrappers over primitive types and our integer
          class is an example of this type. We do want our
```

```
        users to be able to initialise the user defined
        object of Integer with the primitive type. If
        that's the case, omit the explicit keyword used
        by the Integer class.
65  }
66
```