Casting in C++

- **Implicit Conversion:** This occurs when assigning a variable the value of which is not expected. For example, if I declare an integer variable and assign it a value such as 10.10, a cast occurs. The .10 is removed and now this variable now holds the value of just 10 which is a valid integer value. I do not need to tell the compiler what to do as it knows even though I have not directly expressed what I want to happen. This is regular C style casting.
- **Regular C Style Cast:** This is casting an object to a pointer of some class and storing it in a pointer of that class i.e. Class *myObject = (Class *)castObject.
- **Dynamic_Cast:** This prevents the developer from making incorrect castings. This allows objects to be cast to a generalised type in a hierarchy. This is called upcasting. However, the developer may want to cast this object to another object below where it is in the hierarchy. This is called downcasting. The good thing about this is that if the developer is trying to force an invalid cast, no exception is thrown but the operation returns null to indicate that it didn't work. If it is valid and works, a pointer is returned.
- **Static_Cast:** This is what we do when we want to get the full precision of a number. If we take the following code:

  double total = 10/20;

  Total will hold the value of 0. This is not what we want. We can add a cast to it to get the full precision. This is shown below:

  double total = (double)10/20;

  This now holds the value of 0.5. Exactly what we wanted. This is using static_cast.

- **Reinterpret_Cast:** This gives us the ability to cast unrelated objects to one another. This is done by making it a pointer of that class the developer is trying to cast it to. This should not be used as it can cause unpredictable behaviour in a programme.

Daniel Buckley
X00109141