```cpp
//// Lecture 53: L-value and R-values (C++11)
#include <iostream>

//Returns by Value: hence it returns an r-value
int Add(int x, int y) {
    return x + y;
}
//Return by Reference: hence it returns an l-value
int & Transform(int &x) {
    x *= x;
    return x;
}
int main() {
    //x,y and z are L-values & 5, 10 and 8 are r-values
    // Because they are temporary values, they cannot
      come on the left
    // hand side of the assignment operator.
    int x = 5;
    int y = 10;
    int z = 8;

    // Some values will persist beyond the expression
      - these are L-values.
    //Expression returns l-value
    // This is an L-value, so it can appear on the
      LEFT HAND SIDE of the assignment op
    // and we can assign a value to it.
    ++x = 6;
    // Expression (x+y)*z returns an r-value
    // so this means that this line of code will be
      lost, when the next line of code is executed.
    // Some values don't persist beyond the
      expression. (L-values)
    int result = (x+y) * z;

    Transform(y) = 3; // because the function returns
      by reference, the function returns an L-value,
      which has a name, and hence it can be placed on
      the left hand side of the operator.


```

```
35        /// R-value references (C++11)
36        int && r1 = 10;
37        int && r2 = Add(5,8);
38        // Recall that r-value references only can bind
            strictly to temporaries, and we cannot bind r-
            value reference to that L-value.
39        // int && r3 = x; --> ERROR CODE
40
41
42        /// L-value References (C++11) - or reference
            variables (Basically) in C++
43        int &ref = x; // binds to L-value
44        int &ref2 = Transform(y); // binds to a function
            that return by reference.
45        // Remember it can never bind to a temporary
46        // int &ref3 = 10; --> ERROR CODE
47
48    }
49
50    // Purpose of L & R-value references?
51    // They allow us to detect temporaries in expressions,
        so we can write functions that overload based on R-
        values and L-value references.
52
53    // Consider the following example without an R-value
        reference
54    // L-value reference
55    void Print(int &x) {
56        std::cout << "Print(int&)" << std::endl;
57    }
58    // Constant L-value reference
59    void Print(const int &x) {
60        std::cout << "Print(const int&)" << std::endl;
61
62    }
63
64    int main()
65    {
66        int x = 10;
67        Print(x); // x is an l-value, so the l-value
            reference will be invoked.
68
```

```
69      // If we invoke print() function with an R-value,
        the constant reference function will be invoked.
70      Print(3); // will invoke the const int since 3 is
        a temporary.
71  }
72
73
74  // Now consider the example with the R-value reference
75  void Print(int &x) {
76      std::cout << "Print(int&)" << std::endl;
77  }
78  void Print(const int &x) {
79      std::cout << "Print(const int&)" << std::endl;
80
81  }
82  void Print(int &&x) {
83      std::cout << "Print(int &&)" << std::endl;
84  }
85
86  int main()
87  {
88      int x = 10;
89      Print (x); // same
90      Print(3); // will invoke the R-value reference
91      // this can be used to detect temporaries in
        expressions that use user defined objects, and
        this fact can be used to implement move
        semantics which is a faster way of copying
        temporary objects.
92  }
93
```