*Exercise 6: L&R values (lvalue\_rvalue.cc)*

***Explained***

* **This exercise has us explore inheritance as it pertains to const virtual functions, copied objects, and transferred objects.** 
  + First and foremost, let’s remember that lvalues typically refer to data that is stored in memory (variables, references, and arrays).
  + Rvalues, on the other hand, are temporary things that are destroyed without having to be stored (Ex: 5 or int{5} are both rvalues).
  + Also, “&x” in a function header refers to a parameter being passed by reference (typically an Lvalue).
  + “&&x” in a function header refers to an expected **RVALUE**.
* **This output we see here is as follows:**

Text

Description automatically generated

1. func(int && x) Rvalue ref
   * 5 is merely an Rvalue
2. func(int && x) Rvalue ref
   * 2 + 3 is never stored so it too is an Rvalue
3. func(int && x) Rvalue ref
   * int{5} is never stored so it too is an Rvalue
4. func(int & x) Lvalue ref
   * Our first Lvalue! Why? Because we explicitly created the variable “int x = 5” which is an Lvalue because it is stored in memory.
5. func(int && x) Rvalue ref
   * Dividing an Lvalue (x) by an Rvalue (2) still equals/creates an Rvalue since nothing is explicitly stored in memory.
6. func(int && x) rvalue ref
   * Using the “move()” function essentially “steals the guts” or transfers the resources owned by an object. Here we are transferring the contents of our Lvalue “x” (those contents being the Rvalue “5”) and passing as an argument to a function. Again, nothing is being stored, so this is an Rvalue still.
7. func(int const & x): const Lvalue ref
   * Here we created a new **CONST** Lvalue variable “y” and passed it as an argument to our func(). Const variables are different such that they cannot be modified. Passing this as an argument limits the func() overload choices to the one that can take a const passed by reference. The other const function would not work since const &&x accepts a const Rvalue only.
8. func(int && x) Rvalue ref
   * Here we divide our const Lvalue “y” by the Rvalue “2”. From earlier, dividing an Lvalue by an Rvalue creates/equals an Rvalue since nothing is being stored (Yes, we can divide a const variable because the variable itself is not changing, rather we are producing a new result from it). This newly created Rvalue (that is not const) is passed as an argument resulting in the regular “&&x” function to be called (not the “const &&x”).
9. func(int const && x) const Rvalue ref
   * Finally, here we perform another move operation on our Lvalue “y”. The contents of “y” is the **const** Rvalue “5”. Just like before, transferring the resources essentially means we are “cutting” them and providing them as an argument. There is only one remaining function, that has not yet been called, that accepts a const Rvalue.