lvalue: you can take its address

rvalue: you cannot take its address

Consider:

string one("cute");

const string two("fluffy");

string three() { return "kittens"; }

const string four() { return "are an essential part of a healthy diet"; }

Result:

one; // modifiable lvalue

two; // const lvalue

three(); // modifiable rvalue

four(); // const rvalue

&&:rvalue reference

T&& r = T();

Move constructor:

Since an rvalue is going to die at the end of an expression, you can steal its data. Instead of copying it into another object, you move its data into it.

foo(foo&& other)

{

this->length = other.length;

this->ptr = other.ptr;

other.length = 0;

other.ptr = nullptr;

}

foo f1((foo())); // Move a temporary into f1; temporary becomes "empty"

foo f2 = std::move(f1); // Move f1 into f2; f1 is now "empty”

Example:

void someFunc(Widget w); // someFunc's parameter w is passed by value

Widget wid; // wid is some Widget

someFunc(wid); // in this call to someFunc, w is a copy of wid that's created via //copy construction

someFunc(std::move(wid)); // in this call to SomeFunc, w is a copy of wid that's

// created via move construction

🡺In the first call to **someFunc** above, the argument is **wid**. In the second call, the argument is **std::move(wid)**. In both calls, the parameter is **w**.

🡺parameters are lvalues, but the arguments with which they are initialized may be rvalues or lvalues.