C++ Rvalue References Explained

By Thomas Becker about me contact

Last updated: March 2013

Contents

Introduction

Rvalue references are a feature of C++ that was added with the C++11 standard. What makes rvalue references a bit difficult to grasp is that when you first look at them, it is not clear what their purpose is or what problems they solve. Therefore, I will not jump right in and explain what rvalue references are. Instead, I will start with the problems that are to be solved and then show how rvalue references provide the solution. That way, the definition of rvalue references will appear plausible and natural to you.

Rvalue references solve at least two problems:

1. Implementing move semantics
2. Perfect forwarding

If you are not familiar with these problems, do not worry. Both of them will be explained in detail below. We'll start with move semantics. But before we're ready to go, I need to remind you of what lvalues and rvalues are in C++. Giving a rigorous definition is surprisingly difficult, but the explanation below is good enough for the purpose at hand.

The original definition of lvalues and rvalues from the earliest days of C is as follows: An *lvalue* is an expression `e` that may appear on the left or on the right hand side of an assignment, whereas an *rvalue* is an expression that can only appear on the right hand side of an assignment. For example,

```
int a = 42;
int b = 43;

// a and b are both l-values:
a = b; // ok
b = a; // ok
a = a * b; // ok

// a * b is an rvalue:
int c = a * b; // ok, rvalue on right hand side of assignment
a * b = 42; // error, rvalue on left hand side of assignment
```

In C++, this is still useful as a first, intuitive approach to lvalues and rvalues. However, C++ with its user-defined types has introduced some subtleties regarding modifiability and assignability that cause this definition to be incorrect. There is no need for us to go further into this. Here is an alternate definition which, although it can still be argued with, will put you in a position to tackle rvalue references: An *lvalue* is an expression that refers to a memory location and allows us to take the address of that memory location via the & operator. An *rvalue* is an expression that is not an lvalue. Examples are:

```
// lvalues:
//
int i = 42;
i = 43; // ok, i is an lvalue
int* p = &i; // ok, i is an lvalue
int& foo();
foo() = 42; // ok, foo() is an lvalue
int* p1 = &foo(); // ok, foo() is an lvalue

// rvalues:
//
int foobar();
int j = 0;
j = foobar(); // ok, foobar() is an rvalue
int* p2 = &foobar(); // error, cannot take the address of an rvalue
j = 42; // ok, 42 is an rvalue
```

If you are interested in a rigorous definition of rvalues and lvalues, a good place to start is Mikael Kilpeläinen's ACCU article on the subject.

1 2 3 4 5 6 7 8 9 10 11

FOLLOW ME ON twitter