

What is rvalue reference?

Arash
November 1st

CodenGrow

lvalue and rvalue definition in C

- lvalue can appear in the left and right side of the assignment
- rvalue always appear in the right side of the assignment

`int a = 42; // a is lvalue but 42 is rvalue`

`a = b; // a and b are both lvalue`

`int c = a * b; // a*b is rvalue and c is lvalue`

lvalue and rvalue definition in C++

- lvalue refers to a memory location and allows us to take the address via & operator
- rvalue is the opposite

```
int i = 42;  
int * p = &i; // i is an lvalue  
int & foo() {return &i;}  
foo() = 43;  
int * p1 = &foo(); // foo() is  
an lvalue because you can  
get its address
```

```
int foobar();  
int j = foobar() // foobar is an  
rvalue  
j = 42 // 42 is an lvalue  
foobar() = 43 //ERROR
```

Why returning lvalue in a function?

- The ability of C++ to return lvalues from functions is important for implementing some overloaded operators.
- Example: Overloading the brackets operator []
 - `std::map<int, float> mymap;`
 - `mymap[10] = 5.6;`

Conversion between lvalue and rvalue

- lvalue to rvalue:
 - `int c = a + b;` //a and b are both lvalue but the operator “+” returns “a + b” which is an rvalue.
- rvalue to lvalue:
 - **int** arr[] = {1, 2};
 - **int*** p = &arr[0];
 - `*(p + 1) = 10;` // “p + 1” is an rvalue but operator “*” makes it lvalue

rvalue reference by example (1)

- Consider the following class

```
class Intvec {  
    public:  
        Intvec(size_t num = 0) : m_size(num), m_data(new int[m_size]) {  
            log("constructor");  
        }  
        ~Intvec() {  
            log("destructor");  
            if (m_data) { delete[] m_data; m_data = 0; }  
        }  
        Intvec(const Intvec& other) : m_size(other.m_size), m_data(new int[m_size]) {  
            log("copy constructor");  
            for (size_t i = 0; i < m_size; ++i)  
                m_data[i] = other.m_data[i];  
        }  
        Intvec& operator=(const Intvec& other) {  
            log("copy assignment operator");  
            Intvec tmp(other);  
            std::swap(m_size, tmp.m_size);  
            std::swap(m_data, tmp.m_data);  
            return *this;  
        }  
    private:  
        void log(const char* msg) { cout << "[" << this << "]" " << msg << "\n"; }  
        size_t m_size; int* m_data;  
};
```

rvalue reference by example (2)

- Now suppose the following assignment is taking place.

```
Intvec v1(20);  
Intvec v2;  
cout << "assigning lvalue...\n";  
v2 = v1;  
cout << "ended assigning lvalue...\n";
```

- How many functions from class Intvec are called?

rvalue reference by example (3)

- copy assignment operator
- copy constructor
- destructor

```
Intvec& operator=(const Intvec& other) {  
    log("copy assignment operator");  
    Intvec tmp(other);  
    std::swap(m_size, tmp.m_size);  
    std::swap(m_data, tmp.m_data);  
    return *this;  
}
```


rvalue reference by example (4)

- Now suppose the following assignment is taking place.

```
Intvec v2;  
cout << "assigning lvalue...\n";  
v2 = v1(20);  
cout << "ended assigning lvalue...\n";
```

- How many functions from class Intvec are called?

rvalue reference by example (5)

- constructor : **v1(20);**
- copy assignment operator
- copy constructor : **Intvec tmp(other);**
- destructor
- destructor : **~v1();**

```
Intvec& operator=(const Intvec& other) {  
    log("copy assignment operator");  
    Intvec tmp(other);  
    std::swap(m_size, tmp.m_size);  
    std::swap(m_data, tmp.m_data);  
    return *this;  
}
```

rvalue reference by example (5)

- What if in assignment we knew that the parameter is temporary?

```
Intvec& operator=(<tell me compiler if other is an lvalue or an rvalue> other) {
```

```
    // if other is an rvalue then i don't need to create tmp
```

```
    //because other itself is temporary
```

```
    //Intvec tmp(other);
```

```
    //std::swap(m_size, tmp.m_size);
```

```
    //std::swap(m_data, tmp.m_data);
```

```
    std::swap(m_size, other.m_size);
```

```
    std::swap(m_data, other.m_data);
```

```
    return *this;
```

```
}
```

rvalue reference by example (6)

- And this is why rvalue reference is invented!

```
Intvec& operator=(Intvec && other) {  
    log("move assignment operator");  
    std::swap(m_size, other.m_size);  
    std::swap(m_data, other.m_data);  
  
    return *this;  
}
```

rvalue reference by example (7)

- Now suppose the following assignment is taking place.

```
Intvec v2;
```

```
cout << "assigning lvalue...\n";
```

```
v2 = v1(20);
```

```
cout << "ended assigning lvalue...\n";
```

- How many functions from class Intvec are called?

rvalue reference by example (8)

- constructor : **v1(20);**
- move assignment operator
- destructor : **~v1();**

```
Intvec& operator=(Intvec && other) {  
    log("move assignment operator");  
    std::swap(m_size, other.m_size);  
    std::swap(m_data, other.m_data);  
  
    return *this;  
}
```

std::move

- std::move transforms an lvalue to rvalue.

```
void rval(int && m) {  
    m = m + 1;  
    cout << "rval: " << m << endl;  
}  
rval(5); //it works because 5 is a rvalue  
int t = 42;  
rval(t); //it does not work because t is a lvalue  
rvalue(move(t)); //it work because move transforms t to rvalue  
  
cout << "t=" << t << endl; // now t has changed to 43!
```

Example of move constructor

```
class ArrayWrapper
{
public:
    // default constructor produces
    // a moderately sized array
    ArrayWrapper ()
        : _p_vals( new int[ 64 ] )
        , _metadata( 64, "ArrayWrapper" )
    {}

    ArrayWrapper (int n)
        : _p_vals( new int[ n ] )
        , _metadata( n, "ArrayWrapper" )
    {}

    // move constructor
    ArrayWrapper (ArrayWrapper&& other)
        : _p_vals( other._p_vals )
        , _metadata( move(other._metadata))
    {
        other._p_vals = NULL;
        other._size = 0;
    }
```

```
    // copy constructor
    ArrayWrapper (const ArrayWrapper& other)
        : _p_vals( new
int[ other._metadata.getSize() ] )
        , _metadata( other._metadata )
    {
        for ( int i = 0; i < _metadata.getSize(); +
+i )
        {
            _p_vals[ i ] = other._p_vals[ i ];
        }
    }

    ~ArrayWrapper ()
    {
        delete [] _p_vals;
    }

public:
    int *_p_vals;
    Metadata _metadata;
};
```


Sources

- [1] <http://eli.thegreenplace.net/2011/12/15/understanding-lvalues-and-rvalues-in-c-and-c>
- [2] <http://www.cprogramming.com/c++11/rvalue-references-and-move-semantics-in-c++11.html>