Value Categories, Constructors and Constness

Ulises Tirado Zatarain

Jun, 2021

Introduction

The C++17 standard (not only C++17) defines expression value categories as follows:

- A glvalue is an expression whose evaluation determines the identity of an object, bit-field, or function.
- A prvalue is an expression whose evaluation initializes an object or a bit-field, or computes the value of the operand of an operator, as specified by the context in which it appears.
- An xvalue is a glvalue that denotes an object or bit-field whose resources can be reused (usually because it is near the end of its lifetime). Example: Certain kinds of expressions involving rvalue references (8.3.2) yield xvalues, such as a call to a function whose return type is an rvalue reference or a cast to an rvalue reference type.

Introduction

- An lvalue is a glvalue that is not an xvalue.
- An rvalue is a prvalue or an xvalue.

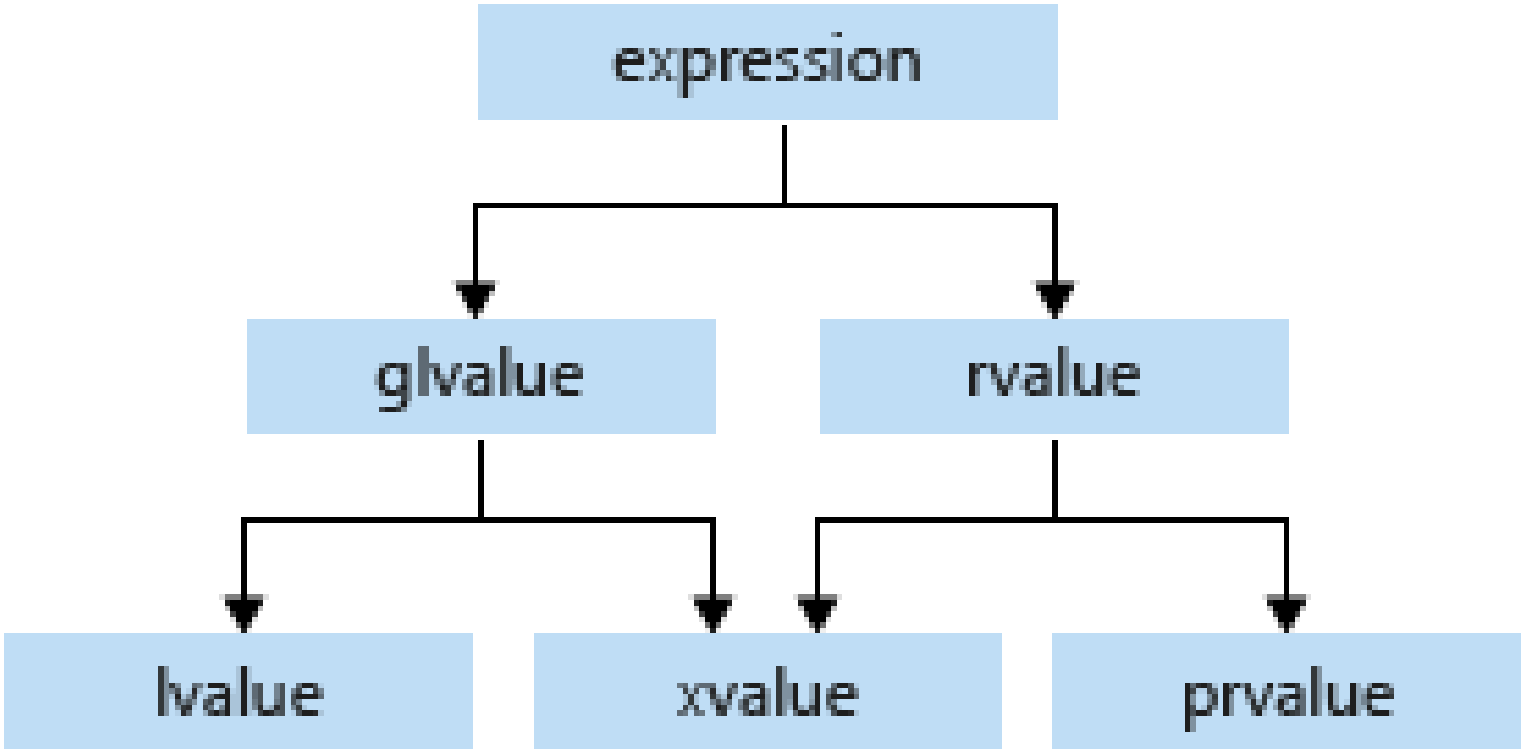

Figure 1: Value categories

Example: Lvalues and Rvalues

```
int i, j, *p;

// Correct usage: the variable i is an lvalue
// and the literal 7 is a prvalue.
i = 7;

// Incorrect usage: The left operand must be
// an lvalue (C2106). `j * 4` is a prvalue.
7 = i; // C2106
j * 4 = 7; // C2106

// Correct usage: the dereferenced pointer is
// an lvalue.
*p = i;
```

Example: Lvalues and Rvalues

```
// Correct usage: the conditional operator
// returns an lvalue.
((i < 3) ? i : j) = 7;

// Incorrect usage: the constant ci is a
// non-modifiable lvalue (C3892).
const int ci = 7;
ci = 9; // C3892
```

Constructors and destructors

```
class A {
public:
    // Default constructor
    A();
    // Following constructor may cause an error
    A(int x = 10, char c = 'a');
    // Copy constructor
    A(const A& copy_from);
    // Since C++11: Move constructor
    A(const A&& move_from);
    // Assignament operator
    A& operator=(const A& copy_from);
    // Destructor
    /*virtual*/ ~A();
};
```

Delete constructors (since C++11)

```
class A {
public:
    // Delete constructor
    A() = delete;
    // Delete copy constructor
    A(const A&) = delete;
    // Delete move constructor
    A(const A&&) = delete;
    // Delete assignament operator
    A& operator=(const A&) = delete;
};
```

Default behaviour for constructors (since C++11)

```
class A {
public:
    // Defaulting default constructor
    A() = default;
    // Defaulting copy constructor
    A(const A&) = default;
    // Defaulting move constructor
    A(const A&&) = default;
    // Defaulting assignament operator
    A& operator=(const A&) = default;
};
```

Constness

```
class pair {
    int value;
    std::string name;
public:
    pair(/*const*/ std::string& label, int x):
        name(label), value(x) { }

    std::string get_value() const {
        return value;
    }

    void set_value(int x) {
        value = x;
    }
};
```

What would happen on push, push_back, emplace, etc?

```
#include <iostream>
using namespace std;
class my_class {
    string label;
public:
    my_class(const string& name): label(name) {
        cout << "label = " << label << endl;
    }

    my_class(const my_class& mc) {
        cout << "Copying: " << mc.label << endl;
    }

    my_class(const my_class&& mc) {
        cout << "Moving: " << mc.label << endl;
    }
};
```

```cpp
#include <sstream>
#include <stack>

my_class create_something(int id) {
    std::stringstream name;
    name << "Number " << id;
    my_class object(name.str());
    // maybe do something else
    return object;
}

int main() {
    std::stack<my_class> stack;
    my_class hello("hello");
    stack.push(hello);
    stack.emplace(create_something(4));
    return 0;
}
```

References

- Value categories
- LValues and RValues in Microsoft Visual C++
- Explicitly defaulted and deleted functions
- All about emplace in C++
- Understanding lvalues and rvalues
- Assertions related with constructors:
  - Constructible
  - Default constructible
  - Copy constructible
  - Move construtible