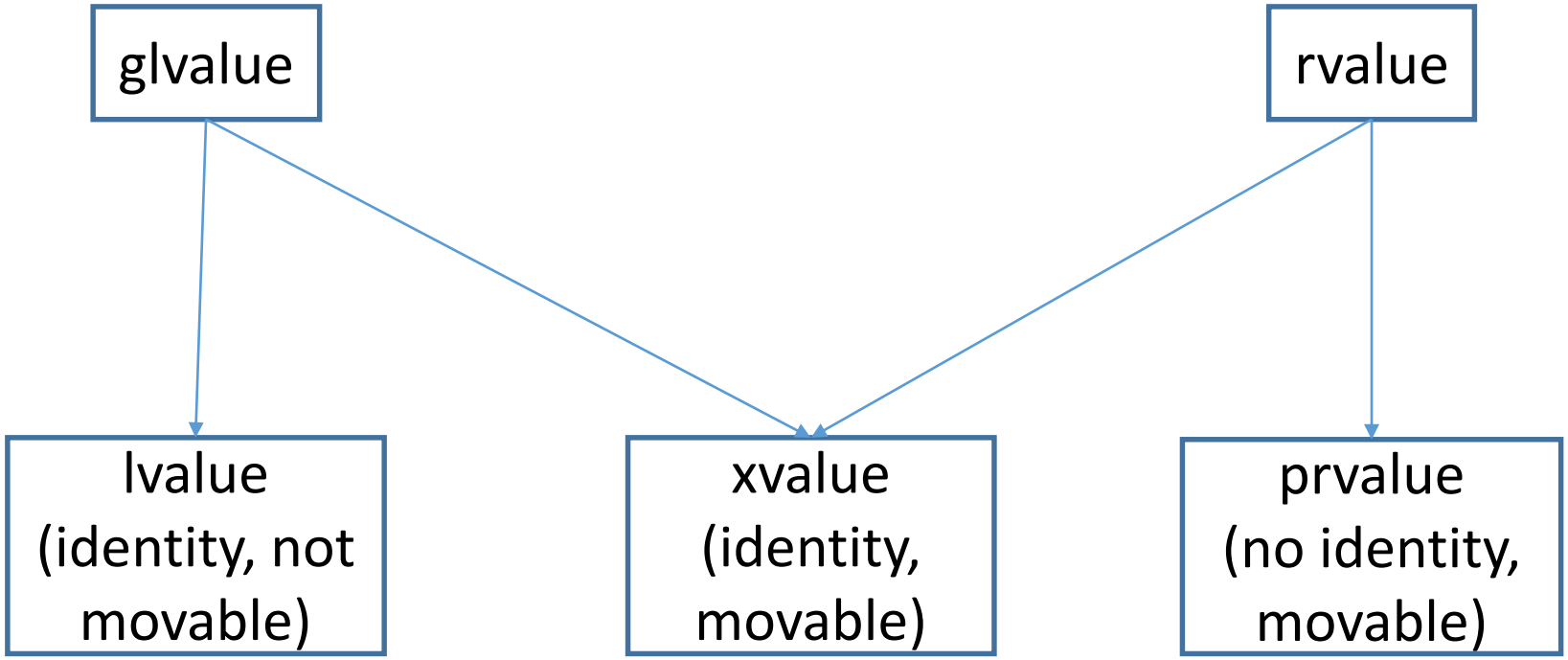
Типы выражений в C++11


glvalue
rvalue
lvalue
(identity, not
movable)
xvalue
(identity,
movable)
prvalue
(no identity,
movable)

glvalue

- A glvalue may be implicitly converted to prvalue with lvalue-to-rvalue, array-to-pointer, or function-to-pointer implicit conversion.
- A glvalue may be polymorphic: the dynamic type of the object it identifies is not necessarily the static type of the expression.
- A glvalue can have incomplete type, where permitted by the expression.

lvalue

- Same as glvalue.
- Address of an lvalue may be taken: **&++i** and **&std::endl** are valid expressions.
- A modifiable lvalue may be used as the left-hand operand of the built-in assignment operator.
- An lvalue may be used to initialize an lvalue reference; this associates a new name with the object identified by the expression.

lvalue examples

- the name of a variable or function in scope, regardless of type, such as **std::cin** or **std::endl**. Even if the variable's type is rvalue reference, the expression consisting of its name is an lvalue expression;
- a function call or an overloaded operator expression of lvalue reference return type, such as **std::getline(std::cin, str)**, **std::cout << 1**, **str1 = str2** or **++it;**
- **a = b**, **a += b**, **a %= b**, and all other built-in assignment and compound assignment expressions;
- **++a** and **--a**, the built-in pre-increment and pre-decrement expressions;
- ***p**, the built-in indirection expression;
- **a[n]** and **p[n]**, the built-in subscript expressions, except where a is an array rvalue (since C++11);
- **a.m**, the member of object expression, except where m is a member enumerator or non-static member function, or where a is an rvalue and m is a non-static data member of non-reference type;
- **p->m**, the built-in member of pointer expression, except where m is a member enumerator or non-static member function;
- **a.*mp**, the pointer to member of object expression, where a is an lvalue and mp is a pointer to data member;
- **p->*mp**, the built-in pointer to member of pointer expression, where mp is a pointer to data member;
- **a, b**, the built-in comma expression, where **b** is an lvalue;
- a string literal, such as **"Hello, world!";**
- a cast expression to lvalue reference type, such as **static_cast<int&>(x);**
- a cast expression to rvalue reference to function type, such as **static_cast<void (&&)(int)>(x);**
- a function call or overloaded operator expression of rvalue reference to function return type.

rvalue

- Address of an rvalue may not be taken: **&int()**, **&i++**, **&42**, and **&std::move(val)** are invalid.
- An rvalue can't be used as the left-hand operand of the built-in assignment or compound assignment operator.
- An rvalue may be used to initialize a const lvalue reference, in which case the lifetime of the object identified by the rvalue is extended until the scope of the reference ends.
- An rvalue may be used to initialize an rvalue reference, in which case the lifetime of the object identified by the rvalue is extended until the scope of the reference ends.
- When used as a function argument and when two overloads of the function are available, one taking rvalue reference parameter and the other taking lvalue reference to const parameter, rvalues bind to the rvalue reference overload (thus, if both copy and move constructors are available, rvalue arguments invoke the move constructor, and likewise with copy and move assignment operators).

prvalue

- Same as rvalue
- A prvalue cannot be polymorphic: the dynamic type of the object it identifies is always the type of the expression.
- A non-class prvalue cannot be cv-qualified.
- A prvalue cannot have incomplete type (except for type void, see below, or when used in decltype specifier).

prvalue examples

- a literal (except for string literal), such as **42**, **true** or **nullptr**;
- a function call or an overloaded operator expression of non-reference return type, such as **str.substr(1, 2)**, **str1 + str2** or **it++**;
- **a++** and **a--**, the built-in post-increment and post-decrement expressions;
- **a + b**, **a % b**, **a & b**, **a << b**, and all other built-in arithmetic expressions;
- **a && b**, **a || b**, **~a**, the built-in logical expressions;
- **a < b**, **a == b**, **a >= b**, and all other built-in comparison expressions;
- **&a**, the built-in address-of expression;
- **p->m**, the built-in member of pointer expression, where **m** is a member enumerator or non-static member function;
- **p->*mp**, the built-in pointer to member of pointer expression, where **mp** is a pointer to member function;
- **a, b**, the built-in comma expression, where **b** is an rvalue;
- **a ? b : c**, the ternary conditional expression for some **a**, **b**, and **c**;
- a cast expression to non-reference type, such as **static_cast<double>(x)**, **std::string{}**, or **(int)42**;
- a lambda expression, such as **[](int x){ return x * x; }**.

xvalue

Properties:

- Same as rvalue.
- Same as glvalue.

Examples:
•a function call or overloaded operator expression of rvalue reference to object return type, such as **std::move(x)**;
•**a[n]**, the built-in subscript expression, where **a** is an array rvalue;
•**a.m**, the member of object expression, where **a** is an rvalue and **m** is a non-static data member of non-reference type;
•**a.*mp**, the pointer to member of object expression, where **a** is an rvalue and **mp** is a pointer to data member;
•**a ? b : c**, the ternary conditional expression for some **a**, **b**, and **c**;
•a cast expression to rvalue reference to object type, such as **static_cast<char&&>(x)**.