# Rvalue and Lvalue References

In this lesson, we will discuss rvalue and lvalue references.

WE'LL COVER THE FOLLOWING

## Rvalues and Lvalues

**Rvalues** are

- Temporary objects
- Objects without a name
- Objects from which we can not get the address

If one of these characteristics holds for an object, it is an rvalue. On the other hand, values with a name and an address are **lvalues**.

- Lvalues can be on the left side of an assignment operator.
- Rvalues can only be on the right side of an assignment operator.

A few examples of rvalues:

```
int five= 5;
std::string a= std::string("Rvalue");
std::string b= std::string("R") +  std::string("value");
std::string c= a + b;
std::string d= std::move(b);
```

As previously stated, Rvalues are on the right side of an assignment. The value

As previously stated, Rvalues are on the right side of an assignment. The value 5 and the constructor call `std::string("Rvalue")` are rvalues since we cannot

determine either the address or the name of the value 5. The same holds for the addition of the rvalues in the expression `std::string("R") + std::string("value")`.

The addition of the two strings `a + b` is notable. Both strings are lvalues, but the addition creates a temporary object. A special use case is `std::move(b)`. The new C++11 function converts the type of the lvalue `b` into an rvalue reference.

## A Few Exceptions

Earlier, we stated that rvalues are on the right side of an assignment, and lvalues can be on the left side of an assignment. It is important to note that this is not always true:

```
const int five= 5;
five= 6;
```

Even though the variable `five` is an lvalue, it is also a constant, so it cannot be used on the left side of an assignment operator.

> Lvalue references are declared by one `&` symbol. Rvalue references are declared by two `&&` symbols. Lvalues can be bound to lvalue references, and rvalues can be bound to rvalue references or constant lvalue references.

```
MyData myData;

MyData& lvalueRef(myData);

MyData&& rvalueRef(MyData());
const MyData& constLValueRef(MyData());
```

> The binding of rvalues to rvalues references has higher priority.

## Rvalue References: Applications

# Rvalue References: Applications #

## Move Semantic #

- Cheap moving of objects instead of expensive copying.
- No memory allocation and deallocation.
- Non-copyable but moveable objects can be transferred by value.

We will discuss this concept in more detail in a later lesson.

## Perfect Forwarding #

- Forward an object without changing the value categories.

We will discuss this concept in more detail in a later lesson.

---

Let’s take a look at an example of this topic in the next lesson.