# Rvalues and Lvalues

Here, we'll discuss the properties of rvalues, lvalues, and their references.

## Rvalues vs. Lvalues #

Rvalues are

- temporary objects.
- objects without a name.
- objects from which we cannot get an address.
- always on the right side of an assignment operation.

The rest are lvalues. They can only be on the left side of an assignment operator.

```cpp
int lValue = 1998; // 1998 is an rvalue
lvalue = 2011;
const int lValue2 = 2011;
lvalue2 = 2011; // ERROR

int defInt = int{};
int res = 2000 + 11;
auto func = []{std::cout << "2011" << std::endl;};
```

## Lvalue and Rvalue references #

- Lvalue references are declared by one `&` symbol.

- Rvalue references are declared by two `&&` symbols.

Lvalues can only be bound to lvalue references. However, 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 rvalue references has higher priority.

# Rvalue references: applications #

## Move semantics #

- Cheap moving of objects instead of expensive copying.

- No memory allocation and deallocation.

- Non-copyable but movable objects can be transferred by value.

## Perfect forwarding #

- Forward an object without changing its rvalue/lvalue nature. This helps in function templates.

In the next lesson, we'll study an example of rvalue references.