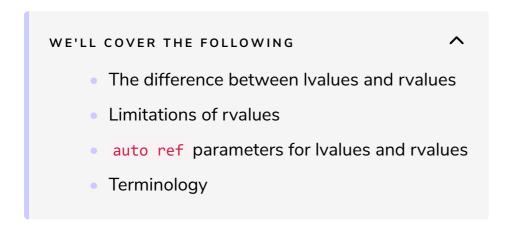
### Lyalues and Ryalues

This lesson explains the difference between the Ivalues and rvalues, limitations of rvalues and how both can be passed as parameters to functions.



### The difference between Ivalues and rvalues #

The value of every expression is classified as either an *lvalue* or an *rvalue*. A simple way of differentiating the two is thinking of lvalues as actual variables (including elements of arrays and associative arrays) and rvalues as temporary results of expressions (including literals).

As a demonstration, the first writeln() expression below uses only values, and the other one uses only rvalues:

```
import std.stdio;
                                                                                        G
void main() {
   int i;
   immutable(int) imm;
   auto arr = [1];
   auto aa = [ 10 : "ten" ];
   /* All of the following arguments are lvalues. */
   writeln(i,
                       // mutable variable
                       // immutable variable
            imm,
                       // array
            arr,
                       // array element
            arr[0],
                       // associative array element
            aa[10]);
   enum message = "hello";
```



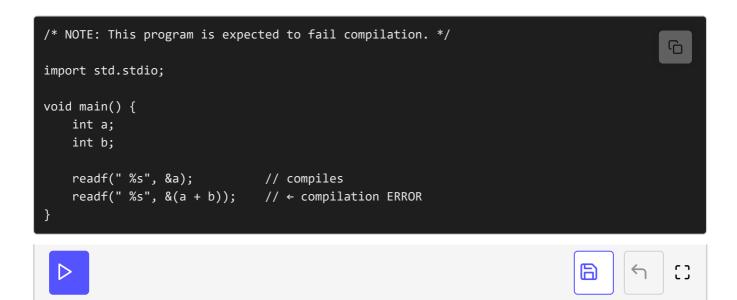
## Limitations of rvalues #

Compared to lvalues, rvalues have the following three limitations.

#### • Rvalues don't have memory addresses

An lvalue has a memory location to which we can refer, while an rvalue does not.

For example, it is not possible to take the address of the rvalue expression a + b in the following program:



Compilation error because rvalues don't have memory addresses

### Rvalues cannot be assigned new values

If mutable, an lvalue can be assigned a new value. This cannot be done with an rvalue:

```
/* NOTE: This program is expected to fail compilation. */
import std.stdio;

void main() {
   int a;
   int b;
   a = 1; // \( \) compiles
   (a + b) = 2; // \( \) compilation ERROR
}
```

Compilation error because rvalues don't have memory addresses

• Rvalues cannot be passed to functions by reference

An Ivalue can be passed to a function that takes a parameter by reference, while an rvalue cannot be:



Compilation error because rvalues don't have memory addresses

## auto ref parameters for Ivalues and rvalues

As mentioned earlier in the chapter, auto ref parameters of function templates can take both lvalues and rvalues.

When the argument is an Ivalue, auto ref means by reference. On the other hand, since rvalues cannot be passed to functions by reference when the argument is an rvalue, it means by copy. For the compiler to generate code

differently in these two distinct cases, the function must be a template.

We will see templates in a later chapter. For now, please accept that the empty parentheses in the highlighted line below make the following definition a function template.

```
import std.stdio;
                                                                                       G
void incrementByTen()(auto ref int value) {
   /* WARNING: The parameter may be a copy if the argument is
    * an rvalue. This means that the following modification
     * may not be observable by the caller. */
   value += 10;
   writeln(value);
}
void main() {
   int a;
   int b;
                        // lvalue; passed by reference
   incrementByTen(a);
   incrementByTen(a + b);
                            // rvalue; copied
```

Using auto ref parameters to accept both Ivalues and rvalues

It is possible to determine whether the parameter is an lvalue or an rvalue by using \_\_traits(isRef) with static if :

```
void incrementByTen()(auto ref int value) {
    static if (__traits(isRef, value)) {
        // 'value' is passed by reference
    } else {
        // 'value' is copied
    }
}
```

```
import std.stdio;

void incrementByTen()(auto ref int value) {
    static if (__traits(isRef, value)) {
        // 'value' is passed by reference
        writeln("value is passed by reference");
    } else {
        // 'value' is copied
        writeln("value is copied");
}
```

```
wittern( value is copied ),
}

void main() {
  int a;
  int b;

  incrementByTen(a);  // lvalue; passed by reference
  incrementByTen(a + b);  // rvalue; copied
}
```

Using auto ref parameters to accept both Ivalues and rvalues

# Terminology #

The names "Ivalue" and "rvalue" do not represent the characteristics of these two kinds of values accurately. The initial letters l and r come from left and right, referring to the left- and the right-hand side expressions of the assignment operator:

- Assuming that it is mutable, an lvalue can be the left-hand expression of an assignment operation.
- An rvalue cannot be the left-hand expression of an assignment operation.

The terms "left value" and "right value" are confusing because in general both lvalues and rvalues can be on either side of an assignment operation:

```
// rvalue 'a + b' on the left, lvalue 'a' on the right:
array[a + b] = a;
```

In the next lesson, you will find a quiz based on the concepts covered in this chapter.