Borrowing and Dereferencing Operators

This lesson discusses the borrowing and dereferencing operator in Rust.

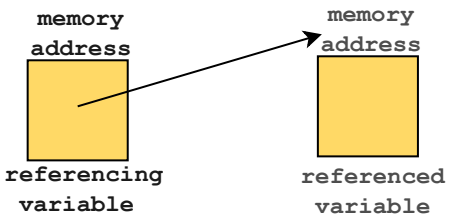
We'll cover the following

Borrowing Operator

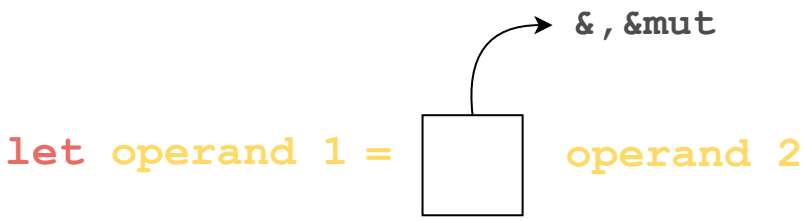
Borrowing means to reference the original data binding or to share the data.

> References are just like pointers in C.

Two variables are involved in a borrowing relationship when the **referenced variable** holds a value that the **referencing variable** borrows. The referencing variable simply points to the memory location of the referenced variable.



The following illustration shows that operand 1 borrows the value of operand 2 using two types of operators:



Types

Borrowing can be of two types:

- **Shared borrowing**

  A piece of data that is shared by single or multiple variables but it cannot be altered

- **Mutable borrowing**

  A piece of data that is shared and altered by a single variable (but the data is inaccessible to other variables at that time)

  The following table summarizes the function of these two types.

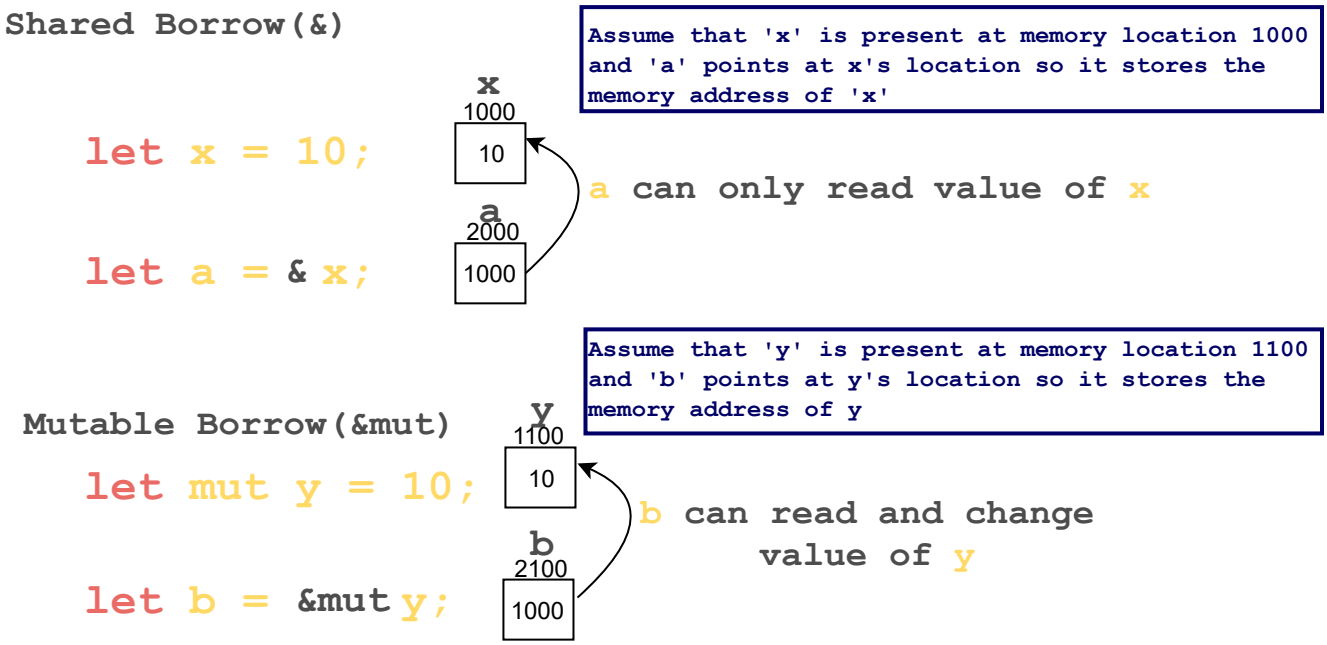
Operator	Operation	Explanation
`operand 1 = & operand 2`	`shared borrow`	`operand1 can read data of another operand2`
`operand 1 = & mut operand 2`	`mutable borrow`	`operand1 can read and alter the data of operand2`

Borrowing Operators

> **Note:** Mutable references(mutable borrow operations) are moved while immutable references(shared borrow operations) are copied.

Example

The following example shows a shared borrow and mutable borrow:



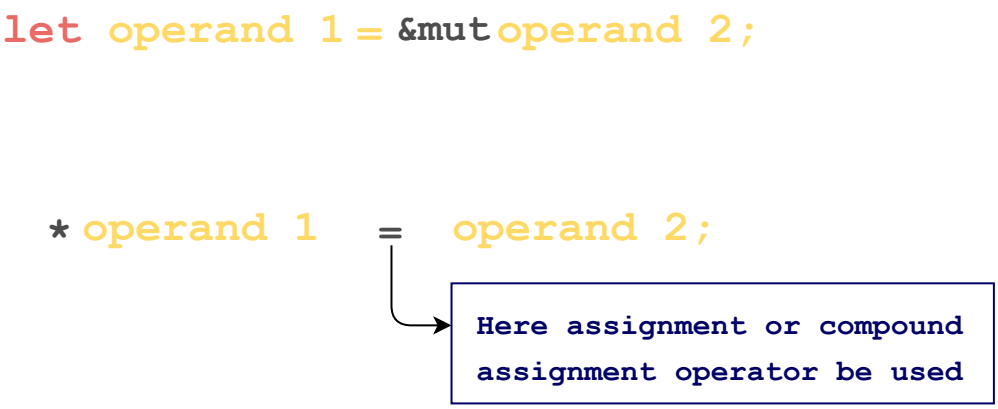

```
fn main() {
    let x = 10;
    let mut y = 13;
    //immutable reference to a variable
    let a = &x;
    println!("Value of a:{}", a);
    println!("Value of x:{}", x); // x value remains the same since it is immutably borrowed
    //mutable reference to a variable
    let b = &mut y;
    println!("Value of b:{}", b);
    println!("Value of y:{}", y); // y value is changed since it is mutably borrowed
}
```

> More details about borrowing will be covered in the last chapter.

Dereferencing Operator

Once you have a mutable reference to a variable, dereferencing is the term used to refer to changing the value of the referenced variable using its address stored in the referring variable.

The following illustration shows that operand 1 mutably borrows the value of operand 2 using `& mut` and then operand 1 dereferences the value of operand 2 using the `*` operator:



Type

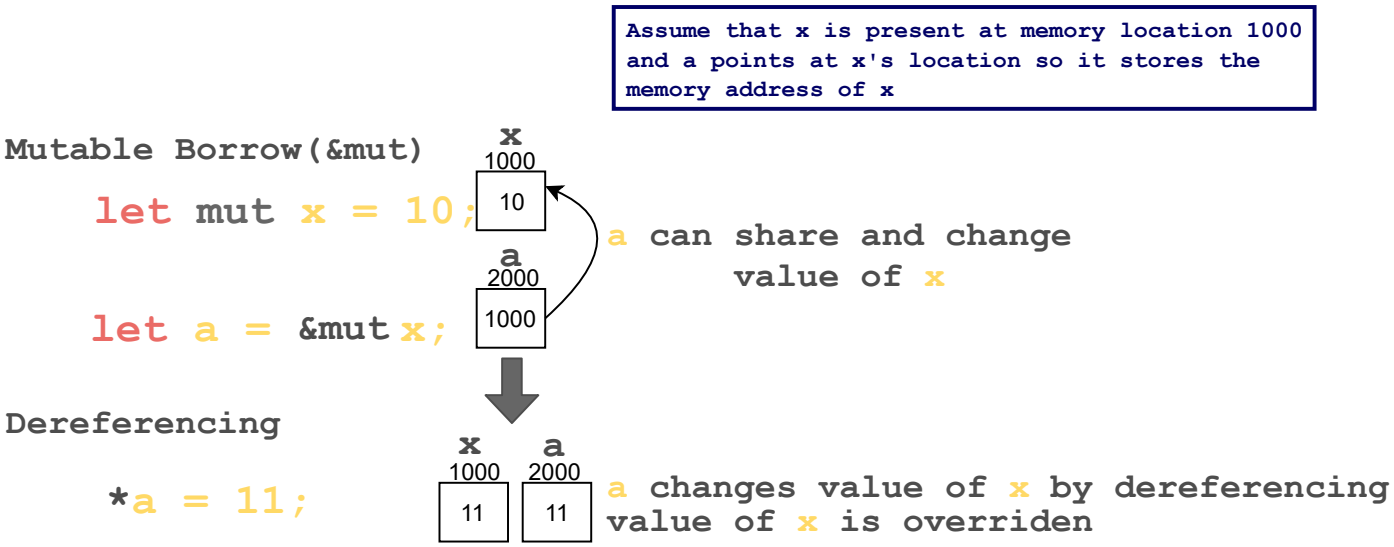
The following table shows the dereferencing operator `*` along with its function.

Operator	Operation	Explanation
* operand 1 = operand 2	dereference a value	point to the value of a mutable borrow variable and can also update that variable value

Dereferencing Operator

Example

The following example shows how to dereference a variable:




```
fn main() {
    //mutable reference to a variable
    let mut x = 10;
    println!("Value of x:{}", x);
    let a = & mut x;
    println!("Value of a:{}", a);
    //dereference a variable
    *a = 11;
```

```
    println!("Value of a:{}", a);
    println!("Value of x:{}", x); // Note that value of x is updated
}
```

Quiz

Test your understanding of borrowing and dereferencing operators in Rust.

Quick Quiz on Borrowing and Dereferencing Operators!

1

A variable can be updated through a dereference operator if it's a

2

What is the output of the following code?

```
fn main() {
      let a = &10;
      let b = &mut 9;
      *b = 12;
      println!("Value of a:{}",a);
      println!("Value of b:{}",b);
}
```

Retake Quiz

Now that you have learned about some of the different operators, let’s learn about the precedence of operators in the next lesson.