

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

main frame

BONUS_PERCENTAGE	10
------------------	----

Heap

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

Employee:new

name	•
salary	5000
sales	5
bonus	0

main frame

john	•
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {
2     // The 'a defines the lifetime of dynamic data
3     name: &'a str,
4     salary: i32,
5     sales: i32,
6     bonus: i32,
7 }
8
9 const BONUS_PERCENTAGE: i32 = 10;
10
11 // salary is borrowed
12 fn get_bonus_percentage(salary: &i32) → i32 {
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;
14     return percentage;
15 }
16
17 // salary is borrowed
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {
19     let bonus_percentage = get_bonus_percentage(salary);
20     let bonus = bonus_percentage * no_of_sales;
21     return bonus;
22 }
23
24 fn main() {
25     // variable is declared as mutable
26     let mut john = Employee {
27         name: &format!("{}", "John"), // explicitly making the value dynamic
28         salary: 5000,
29         sales: 5,
30         bonus: 0,
31     };
32
33     // salary is borrowed while sales is cloned since i32 is a primitive
34     john.bonus = find_employee_bonus(&john.salary, john.sales);
35     println!("Bonus for {} is {}", john.name, john.bonus);
36 }
```

Thread stack

main frame

john	
name	
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

find_employee_bonus

salary	
no_of_sales	5

main frame

john	
name	
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

find_employee_bonus

salary	
no_of_sales	5
bonus_percentage	0

main frame

john	
name	
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) -> i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) -> i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

get_bonus_percentage

salary	
--------	--

find_employee_bonus

salary	
--------	--

no_of_sales	5
-------------	---

bonus_percentage	0
------------------	---

main frame

john	
------	--

name	
------	--

salary	5000
--------	------

sales	5
-------	---

bonus	0
-------	---

BONUS_PERCENTAGE	10
------------------	----

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

get_bonus_percentage

salary	●
percentage	500

find_employee_bonus

salary	●
no_of_sales	5
bonus_percentage	0

main frame

john	
name	●
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

get_bonus_percentage

salary	●
percentage	500
return	500

find_employee_bonus

salary	●
no_of_sales	5
bonus_percentage	0

main frame

john	
name	●
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	
10	

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {
2     // The 'a defines the lifetime of dynamic data
3     name: &'a str,
4     salary: i32,
5     sales: i32,
6     bonus: i32,
7 }
8
9 const BONUS_PERCENTAGE: i32 = 10;
10
11 // salary is borrowed
12 fn get_bonus_percentage(salary: &i32) → i32 {
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;
14     return percentage;
15 }
16
17 // salary is borrowed
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {
19     let bonus_percentage = get_bonus_percentage(salary);
20     let bonus = bonus_percentage * no_of_sales;
21     return bonus;
22 }
23
24 fn main() {
25     // variable is declared as mutable
26     let mut john = Employee {
27         name: &format!("{}", "John"), // explicitly making the value dynamic
28         salary: 5000,
29         sales: 5,
30         bonus: 0,
31     };
32
33     // salary is borrowed while sales is cloned since i32 is a primitive
34     john.bonus = find_employee_bonus(&john.salary, john.sales);
35     println!("Bonus for {} is {}", john.name, john.bonus);
36 }
```

Thread stack

find_employee_bonus

salary	
no_of_sales	5
bonus_percentage	500

main frame

john	
name	
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) -> i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) -> i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

find_employee_bonus

salary	
no_of_sales	5
bonus_percentage	500
bonus	2500

main frame

john	
name	
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

find_employee_bonus

salary	●
no_of_sales	5
bonus_percentage	500
bonus	2500
return	2500

main frame

john	
name	●
salary	5000
sales	5
bonus	0
BONUS_PERCENTAGE	
10	

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {
2     // The 'a defines the lifetime of dynamic data
3     name: &'a str,
4     salary: i32,
5     sales: i32,
6     bonus: i32,
7 }
8
9 const BONUS_PERCENTAGE: i32 = 10;
10
11 // salary is borrowed
12 fn get_bonus_percentage(salary: &i32) -> i32 {
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;
14     return percentage;
15 }
16
17 // salary is borrowed
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) -> i32 {
19     let bonus_percentage = get_bonus_percentage(salary);
20     let bonus = bonus_percentage * no_of_sales;
21     return bonus;
22 }
23
24 fn main() {
25     // variable is declared as mutable
26     let mut john = Employee {
27         name: &format!("{}", "John"), // explicitly making the value dynamic
28         salary: 5000,
29         sales: 5,
30         bonus: 0,
31     };
32
33     // salary is borrowed while sales is cloned since i32 is a primitive
34     john.bonus = find_employee_bonus(&john.salary, john.sales);
35     println!("Bonus for {} is {}", john.name, john.bonus);
36 }
```

Thread stack

main frame

john	
name	
salary	5000
sales	5
bonus	2500
BONUS_PERCENTAGE	10

Heap

Box<"John">

Rust Memory usage

```
1 struct Employee<'a> {  
2     // The 'a defines the lifetime of dynamic data  
3     name: &'a str,  
4     salary: i32,  
5     sales: i32,  
6     bonus: i32,  
7 }  
8  
9 const BONUS_PERCENTAGE: i32 = 10;  
10  
11 // salary is borrowed  
12 fn get_bonus_percentage(salary: &i32) → i32 {  
13     let percentage = (salary * BONUS_PERCENTAGE) / 100;  
14     return percentage;  
15 }  
16  
17 // salary is borrowed  
18 fn find_employee_bonus(salary: &i32, no_of_sales: i32) → i32 {  
19     let bonus_percentage = get_bonus_percentage(salary);  
20     let bonus = bonus_percentage * no_of_sales;  
21     return bonus;  
22 }  
23  
24 fn main() {  
25     // variable is declared as mutable  
26     let mut john = Employee {  
27         name: &format!("{}", "John"), // explicitly making the value dynamic  
28         salary: 5000,  
29         sales: 5,  
30         bonus: 0,  
31     };  
32  
33     // salary is borrowed while sales is cloned since i32 is a primitive  
34     john.bonus = find_employee_bonus(&john.salary, john.sales);  
35     println!("Bonus for {} is {}", john.name, john.bonus);  
36 }
```

Thread stack

Heap