Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

•••

Karla Ramírez Pulido

Macros

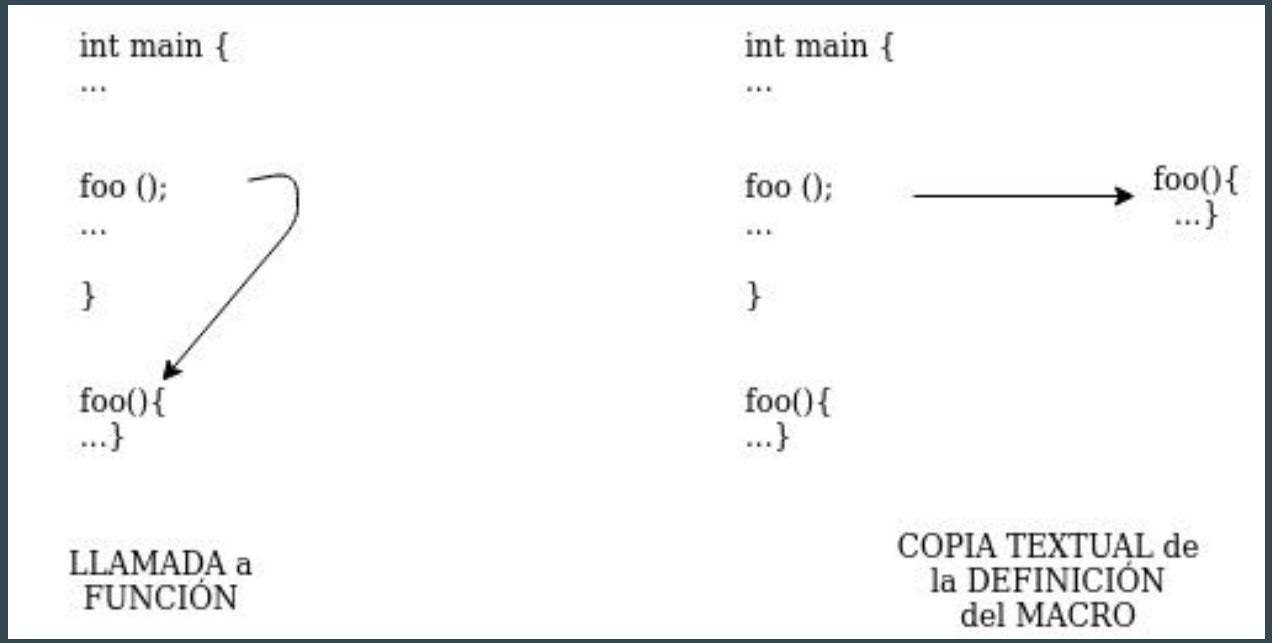
Macro vs Funciones

No es lo mismo

hacer una llamada

a función que

un macro.



Macro en Racket de my-let ques básicamente un with

Si tenemos: {with {var val} body}

Lo podemos sustituir por: { {fun {var} body} val}

Y with y let mantienen la misma semántica sólo es sintaxis distinta podemos escribir:

{let {var val} body} ⇒ { {fun {var} body} val}

Macro en Racket de my-let-1 que es básicamente un with

{let {var val} body} => { {fun {var} body} val}

No puedo usar la primitiva let, entonces le pongo un nuevo nombre a mi macro: **my-let-1**

Entonces el macro sería:

```
(define-syntax my-let-1
  (syntax-rules ( )
    [(my-let-1 (var val) body)
     ((lambda (var) body) val) ] ))
```

Los “...”

Repite el mismo patrón

anterior 0 ó más veces.

(my-let ([x 1] [y 2] [z 3])

(+ x (+ y z)))

Lo que se ejecuta es:

```
(define-syntax my-let
  (syntax-rules ()
    [(my-let ([var val] ···) body)
     ((lambda (var ···) body) val ···)]))
```

((lambda (x y z) (+ x (+ y z)))

1 2 3)

= (+ x (+ y z)) = (+ 1 (+ 2 3)) = 6

Macro: cond ahora cond2

(cond2 (t e1) (**else** e2)) => (if t e1 e2)

```
(define-syntax cond2
  (syntax-rules (else)
    [(cond2 (t e1) (else e2))
     (if t e1 e2)]))
```

Macro: cond ahora cond2

(cond2 (true 1) (else (+ 1 1))

⇒ (if true

1

(+ 1 1))

= 1

```
(define-syntax cond2
  (syntax-rules (else)
    [(cond2 (t e1) (else e2))
     (if t e1 e2)]))
```

Macro my-or2-fun

Si queremos definir un “or” con comportamiento de “if” entonces:

```
(define (my-or2-fun e1 e2)
  (if e1
      e1
      e2))
```

(my-or2-fun e1 e2) ⇒ (if e1 e1 e2)

Sabemos que el OR regresa *true* cuando alguna expresión (e1 y e2) sean TRUE

También sabemos que el IF (si se cumple la *condicional*) se regresa la expresión **e1** (pues es *true*), sino se evalúa **e2.**

Macro my-or2-fun

```
(define (my-or2-fun e1 e2)
   (if e1
      e1
      e2))
```

(my-or2-fun (= 1 0) (+ 2 3))

donde e1 = (= 1 0) y e2 = (+ 2 3)

La expresión se transforma en:

```
(if (= 1 0)
    (= 1 0)
    (+ 2 3))
```

= 5

Macro my-or2-fun

```
(define (my-or2-fun e1 e2)
   (if e1
      e1
      e2))
```

(my-or2-fun (= 0 0) (+ 2 3))

donde e1 = (= 0 0) y e2 = (+ 2 3)

La expresión se transforma en:

```
(if (= 0 0)
    (= 0 0)
    (+ 2 3))
```

= true

Optimización del macro

Evaluamos sólo una única vez “e1”

```
(define (my-or2-fun e1 e2)
  (if e1
      e1
      e2))
```

```
(define-syntax my-or2
  (syntax-rules ()
    [(my-or2 e1 e2)
     (let ([result e1])
       (if result
           result
           e2))]))
```

Optimización del macro

(my-or2 (= 1 1) (+ 1 2))

⇒ (let ([result (= 1 1)])

(if result

result

(+ 1 2)))

```
(define-syntax my-or2
  (syntax-rules ()
    [(my-or2 e1 e2)
     (let ([result e1])
       (if result
           result
           e2))]))
```

⇒ (let ([result **true**])

(if **true** ; cond-expr

true ; then-expr

(+ 1 2) ; else-expr))

= true

¿Dudas?

Gracias