

# Lenguajes de Programación

## Tarea 6

Karla Ramírez Pulido Alan Alexis Martínez López

Semestre 2023-1 **Fecha de inicio:** 11 de mayo 2023  
Facultad de Ciencias UNAM **Fecha de entrega:** 18 de mayo 2023

### Integrantes:

Dania Paula Gongora Ramírez  
Salgado Tirado Diana Laura

### Instrucciones

Resolver los siguientes ejercicios de forma clara y ordenada de acuerdo a los lineamientos de entrega de tareas disponibles en la página del curso.

### Ejercicios

#### 1. Explica con tus propias palabras el concepto y características de una continuación.

Es una técnica que permite utilizar a la pila de ejecución como un valor, lo que permite pausar su ejecución actual y reanudarla más tarde desde ese mismo punto, también ayudan en la gestión de excepciones y en la manipulación del flujo de control.

#### 2. Explica con tus propias palabras el funcionamiento de las primitivas *call/cc* y *let/cc* del lenguaje de programación *Racket* y da un ejemplo de uso de cada una.

-call/cc es una abreviación de call-with-current-continuation, se utiliza para capturar la continuación actual del programa y aplicarla a una expresión dada, En esencia, la función recupera el contexto actual del programa como un objeto y le aplica una función que toma ese objeto como parámetro.

Cuando aplicamos una continuación se aplica, la continuación anterior se elimina, (es decir la que estaba antes de esta nueva), y la nueva continuación será la que se aplique, Por lo tanto, la ejecución del programa continúa hasta que el argumento de la continuación es el valor de retorno de la llamada a la función call/cc.

Ejemplo:

Evaluemos la siguiente expresión: (\* 2 (call/cc (lambda (k) (- 5 (k 8)))))

$= (*\ 2\ (call/cc\ (lambda\ (k)\ (-\ 5\ (k\ 8))))$

--> Aplicando call/cc, se captura la continuación actual y la ligamos con la variable k.

$= (*\ 2\ 8) = 16$

-let/cc opera como una asignación local que utiliza la continuación actual como su identificador, es decir, al utilizar la expresión (let/cc i j), se captura la continuación actual como una función y se asigna a la variable i la continuación que se ha capturado, de esta forma, se puede evaluar la expresión j haciendo uso de la variable i que contiene la continuación capturada.

Ejemplo:

Evaluemos la siguiente expresión:  $(*\ 2\ (let/cc\ k\ (+\ 5\ (k\ (+\ 1\ 3))))$

$= (*\ 2\ (let/cc\ k\ (+\ 5\ (k\ (+\ 1\ 3))))$

→ Aplicamos let/cc, se captura la continuación actual y ligamos la variables k

$= (*\ 2\ (let/cc\ k\ (+\ 5\ (k\ (+\ 1\ 3))))$  con  $k = (+\ 1\ 3) = 8$

$= (*\ 2\ 4) = 8$

**3. Convierte las siguiente función utilizando CPS (*Continuation Passing Style*) y muestra su ejecución utilizando el paso de parámetros por valor, con la instancia entera 5.**

```
(define (factorial n)
  (if (= n 0)
      1
      (+ n (factorial (- n 1)))))
```

Factorial con CPS

```
(define (factorial n)
  (fact/k n (lambda (x) x)))

(define (fact/k n k)
  (if (= n 0)
      (k 1)
      (fact/k (- n 1) (lambda (v) (k (* n v))))))
```

Ejecutamos con la instancia entera 5

$(factorial\ 5)$

$(fact/k\ 5\ (lambda\ (x)\ x))$      $n = 5$  y  $k = (lambda\ (x)\ x)$

$(fact/k\ (-\ n\ 1)\ (lambda\ (x)\ x))$     donde  $n = (-\ 5\ 1) = 4$  y  $k = (lambda\ (x)\ x)$

$(fact/k\ 4\ (\lambda(v)\ ((\lambda(x)\ x)\ (*\ 5\ v))))$     donde  $n = 4$  y  $k = (\lambda(v)\ ((\lambda(x)\ x)\ (*\ 5\ v)))$

$(fact/k\ 3\ (\lambda(v)\ ((\lambda(v)\ ((\lambda(x)\ x)\ (*\ 5\ v)))\ (*\ 4\ v))))$

donde  $n = 3$  y  $k = (\lambda(v)\ ((\lambda(v)\ ((\lambda(x)\ x)\ (*\ 5\ v)))\ (*\ 4\ v)))$

$(fact/k\ 2\ (\lambda(v)\ ((\lambda(v)\ ((\lambda(v)\ ((\lambda(x)\ x)\ (*\ 5\ v)))\ (*\ 4\ v)))\ (*\ 3\ v))))$

donde  $n = 2$  y  $k = (\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v)))$

$(fact/k\ 1\ (\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) (* 2 v)))$

donde  $n = 1$  y  $k = (\lambda(v) ((\lambda(v) (* (\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) (* 2 v)))$

$(fact/k\ 0\ (\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) (* 2 v))) (* 1 v)))$

donde  $n = 0$  y  $k = (\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) (* 2 v))) (* 1 v)))$

como  $(zero? n)$  se cumple, entonces

$(k\ 1)$

Asignamos al parámetro formal “v” su valor que es 1, i.e.  $v = 1$

$((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) (* 2 v))) (* 1\ 1))$

$((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) (* 2 v))) 1)$

$v = 1$

$(\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) (* 2\ 1))$

$((\lambda(v) ((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3 v))) 2)$

$v = 2$

$((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) (* 3\ 2)))$

$((\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4 v))) 6))$

$v = 6$

$(\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (* 4\ 6)))$

$(\lambda(v) ((\lambda(v) ((\lambda(x) x) (* 5 v))) (24)))$

$v = 24$

$((\lambda(x) x) (* 5 24))$

$((\lambda(x) x) (120))$

$x = 120$

Por lo tanto el resultado de factorial 5 es 120


#### 4. Observa la siguiente función del *lenguaje de programación Racket*

```
(let ([fib (lambda (n) (if (or (zero? n) (= n 1)) 1 (+ (fib (- n 1))  
  (fib (- n 2)))))]  
  (fib 3))
```

- a. Prueba la expresión en el intérprete de *Racket* y con base en la respuesta obtenida, explica el proceso que siguió el intérprete para llegar a ésta. Anexa una captura de pantalla del intérprete de *Racket* al probar la expresión.

```
#lang plai
(let ([fib (lambda (n) (if (or (zero? n) (= n 1)) 1 (+ (fib (- n 1))  
  (fib (- n 2)))))]  
  (fib 3))
```

Welcome to [DrRacket](#), version 8.2 [cs].  
Language: **plai**, with **debugging**; memory limit: **128 MB**.

 **fib: unbound identifier in: fib**

>

Se evalúa la lambda con el valor 3, como 3 no es 1 ni 0, pasa directamente a otra condición diferente al caso base, pero en la expresión  $(+ (fib (- n 1)) (fib (- n 2)))$  fib es una variable libre, por lo que se manda error.

- b. Modifica la función usando el Combinador de Punto Fijo Y. Prueba la expresión en el intérprete de *Racket* y con base en la respuesta obtenida, explica el proceso que siguió el intérprete para llegar a ésta. Anexa una captura de pantalla del intérprete de *Racket* al probar la expresión.

```
#lang plai

;;Con punto fijo
(let ([y (λ (f) ((λ (x) (f (x x))) (λ (x) (f (x x))))))]
      [fib (y (λ (f) (λ (n) (if (or (zero? n) (= n 1)) 1 (+ (f (- n 1)) (f (- n 2))))))]
      (fib 3)))

Welcome to DrRacket, version 8.2 [cs].
Language: plai, with debugging; memory limit: 128 MB.
✖ y: unbound identifier in: y
>
```

Se evalúa la lambda con el valor 3, como 3 no es 1 ni 0, pasa directamente a otra condición diferente al caso base, pero en este caso y queda como variable libre, por lo que también se arroja un error.