Junio de 2024

Ejercicios y Trabajo Final

Evidencias

Emiliano Espinoza Perales - 21040337
PROGRAMACIÓN LÓGICA Y FUNCIONAL – 8Y

Programación Lógica y Funcional

Trabajo Final

Resuelve los siguientes ejercicios.

Ejercicio 1. Sobre parejas.

Instrucción	Predicción de resultado
#1	(1 . 2) ✓
#2	1 ✓
#3	2 ✓
#4	(6 . 2) ✓
#5	(3 . "3") ✓
#6	1 ✓
#7	(3 . 4) ✓
#8	2 ✓
#9	(2 . 3) ✓
#10	2 ✓

#6
(1 . 2)
((1 . 2) . 3)
(1 . 2)
→ 1

#7
(3 . 4)
((3 . 4) . 2)
→ (3 . 4)

#8
(3 . 4)
((3 . 4) . 2)
→ 2

#9
(2 . 3)
(1 . (2 . 3))
→ (2 . 3)

#10
(1 . 2)
((1 . 2) . 3)
(1 . 2)
→ 2

Ejercicio 2. Sobre listas.

Instrucción	Predicción de resultado
#1	(1 2 3 4) ✓
#2	(2 3 4) ✓
#3	1 ✓
#4	#t ✓
#5	2 ✓
#6	(3 4) ✓
#7	3 ✓
#8	(4 3 2) ✓
#9	(3 . (1 2 3)) ✗
#10	("Hola" 1) ✓
#11	(1 2) ✓
#12	1 ✓
#13	((1 2 3) 4 5 6) ✓
#14	2
#15	(3 4)

#5
(1 2 3 4)
(2 3 4)
→ 2

#6
(1 2 3 4)
(2 3 4)
→ (3 4)

#7
(1 2 3 4)
(2 3 4)
(3 4)
→ 3

#10
("Hola" 1)
(#t "Hola" 1)
→ ("Hola" 1)

#11
(1 2)
((1 2) 1 2 3 4)
→ (1 2)

#12
((1 2) 1 2)
(1 2)
→ 1

#14
(1 2 3 4)
(2 3 4)
→ 2

#15
(1 2 3 4)
(2 3 4)
→ (3 4)

#16

(1 2 3 4)
(2 3 4)
(3 4)
(4)
4

#16

Ejercicio 3. Sobre listas.

a) Dada la siguiente lista, indica la expresión correcta para que Scheme devuelva 3:

(list 1 2 3 4 5) → (first (rest (rest (list 1 2 3 4 5))))

b) Dada la siguiente lista, indica la expresión correcta para que Scheme devuelva (5).

(list 1 2 3 4 5) → (rest (rest (rest (rest (list 1 2 3 4 5)))))

c) Dada la siguiente lista, indica la expresión correcta para que Scheme devuelva 5.

(list 1 2 3 4 5) → (first (rest (rest (rest (rest (list 1 2 3 4 5))))))

d) Dada la siguiente expresión, ¿qué devuelve Scheme?

(first (rest (rest (list 1 (list 2 3) (list 4 5) 6))))

d)
(1 (2 3) (4 5) 6)
((2 3) (4 5) 6)
((4 5) 6)
→ (4 5) ✓

e) Dada la siguiente expresión, ¿qué devuelve Scheme?

(rest (rest '(1 (2 3) 4 5)))

e)
(1 (2 3) 4 5)
((2 3) 4 5)
→ (4 5) ✓

Ejercicio 4. Predice lo que devolverá Scheme cuando escribas las siguientes expresiones

Instrucción	Predicción de resultado
#1	#t ✓
#2	#\e ✓
#3	"uen" ✓
#4	Error (es para igualdad matemática entre números) ✓
#5	#\n ✓
#6	— ✓
#7	3.14159 ✓
#8	"pi" ✓
#9	9.42477 ✗
#10	-0.6967722719000... ✓
#11	1265 ✓
#12	7
#13	— ✓
#14	19 ✓
#15	#f ✓
#16	3 ✓
#17	16 ✓
#18	6 ✓
#19	16 ✓
#20	7

#10
(* pi pi)
9.8695877281
9.8695877281
-10.56636
-0.6967722719000...
-3.14159
-1.14159
-1.14159
-3.14159
-4.28318
-4.28318
-3.14159
-7.42477
-7.42477
-3.14159
-10.56636

#5
"eno"
→#\n

#9
3.14159 + 3.14159 = 6.28318
3.14159 · 6.28318 = 9.42477 ✗

#11
1200 → ¥
¥ → 1200
1200 + 65
→ 1265

#\A → 65

#13
a=3
b=a+1=3+1=4

#14
12
a+b=7
7+12=19

#12
"EEEEEEE"
→7

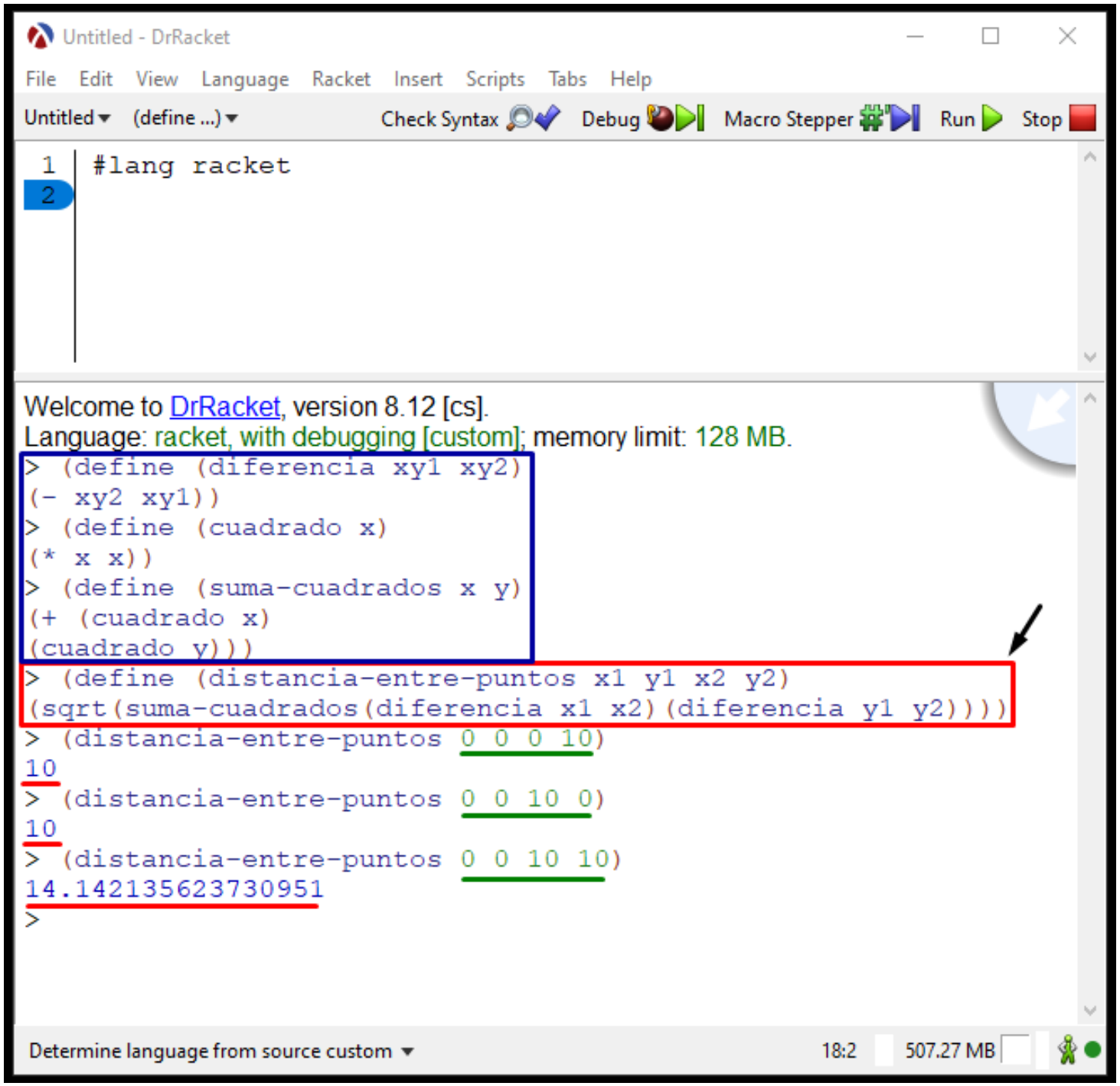
#15
>a b → #f and <b 12 → #t
#f
b a

#17
+6 7 = 13
13+a=16

#19
b
(+a 1)=4
* b 4 = 16

#18
(>b a)
#t
b a
+2 b=6

#20
(<a b)
#t
+ -
(+ a b)=7


Untitled - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
Untitled (define ...) Check Syntax Debug Macro Stepper Run Stop
1 #lang racket
2
Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.
> (define (diferencia xy1 xy2)
(- xy2 xy1))
> (define (cuadrado x)
(* x x))
> (define (suma-cuadrados x y)
(+ (cuadrado x)
(cuadrado y)))
> (define (distancia-entre-puntos x1 y1 x2 y2)
(sqrt(suma-cuadrados(diferencia x1 x2)(diferencia y1 y2))))
> (distancia-entre-puntos 0 0 0 10)
10
> (distancia-entre-puntos 0 0 10 0)
10
> (distancia-entre-puntos 0 0 10 10)
14.142135623730951
>
Determine language from source custom
18:2
507.27 MB

Untitled - DrRacket

File Edit View Language Racket Insert Scripts Tabs Help

Untitled (define ...) Check Syntax Debug Macro Stepper Run Stop

```
1 #lang racket
2
```

Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.

```
> (define (bin-a-dec b3 b2 b1 b0)
    (+ (+ (+ (* b3 (expt 2 3)) (* b2 (expt 2 2))) (* b1 (expt 2 1))) (* b0 (expt 2 0)))
    )
> (bin-a-dec 1 1 1 1)
15
> (bin-a-dec 0 1 1 0)
6
> (bin-a-dec 0 0 1 0)
2
>
```

Untitled - DrRacket

File Edit View Language Racket Insert Scripts Tabs Help

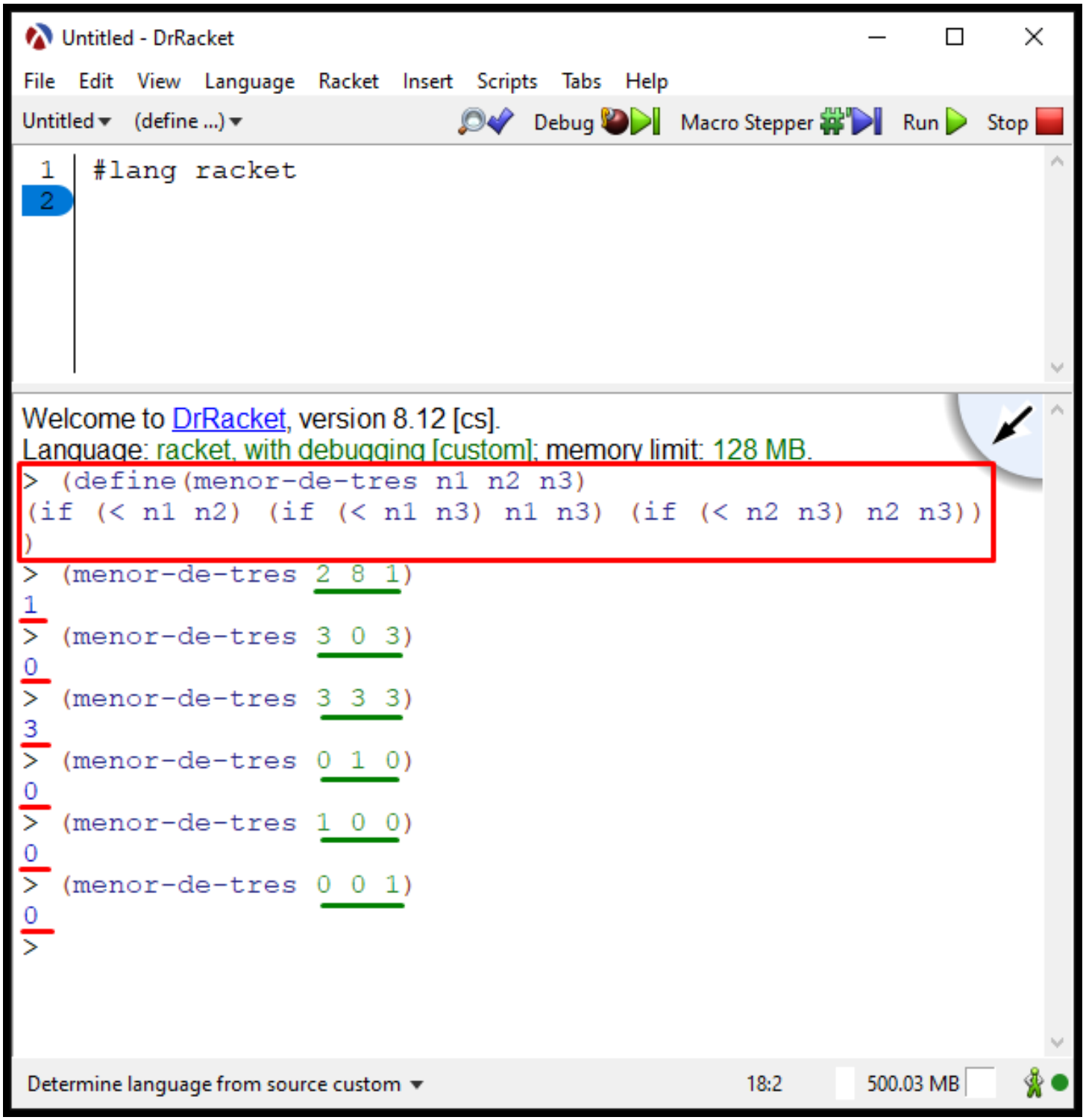
Untitled (define ...) Check Syntax Debug Macro Stepper Run Stop

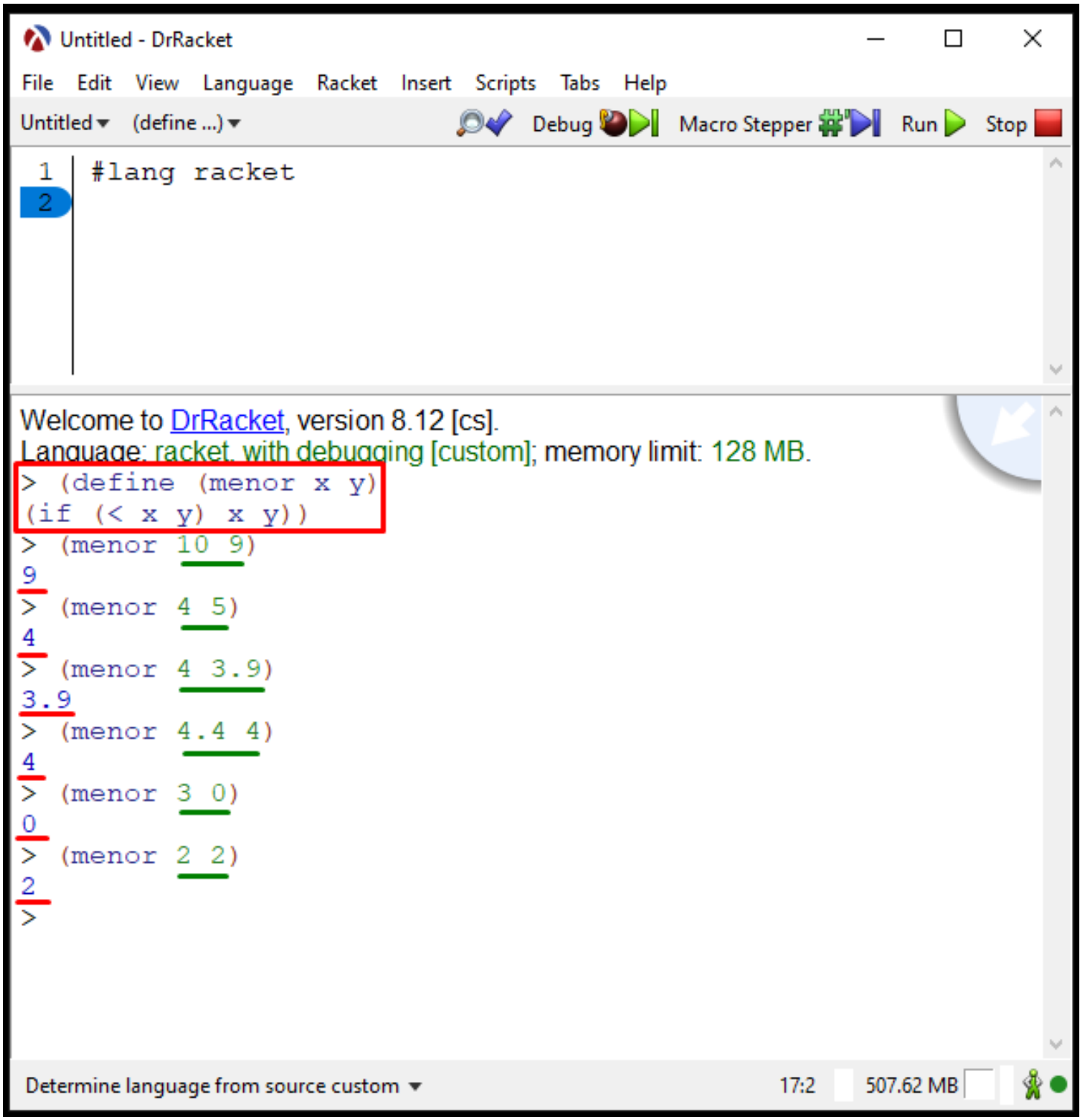
```
1 #lang racket
2
```

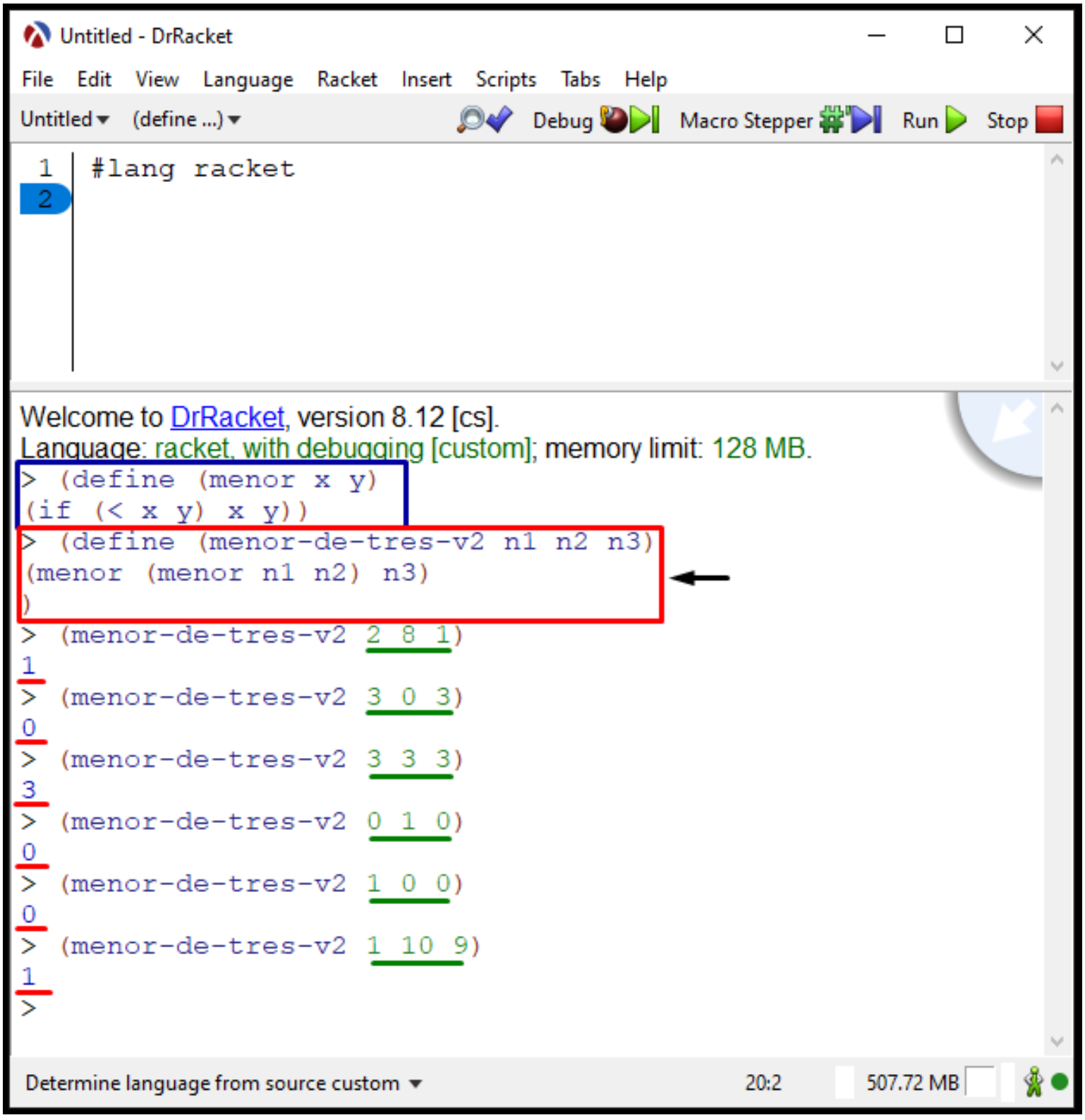
Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.

```
> (define (bin-a-dec b3 b2 b1 b0)
    (+ (+ (+ (* b3 (expt 2 3)) (* b2 (expt 2 2))) (* b1 (expt 2 1))) (* b0 (expt 2 0)))
    )
> (define (bin-a-hex b3 b2 b1 b0)
(integer->char (+ (bin-a-dec b3 b2 b1 b0) (if (< (bin-a-dec b3 b2 b1 b0) 10) 48 55))))
> (bin-a-hex 1 1 1 1)
#\F
> (bin-a-hex 0 1 1 0)
#\6
> (bin-a-hex 1 0 1 0)
#\A
>
```

Determine language from source custom 14:2 498.42 MB


Untitled - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
Untitled (define ...) Debug Macro Stepper Run Stop
1 #lang racket
2
Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.
> (define (menor-de-tres n1 n2 n3)
(if (< n1 n2) (if (< n1 n3) n1 n3) (if (< n2 n3) n2 n3))
)
> (menor-de-tres 2 8 1)
1
> (menor-de-tres 3 0 3)
0
> (menor-de-tres 3 3 3)
3
> (menor-de-tres 0 1 0)
0
> (menor-de-tres 1 0 0)
0
> (menor-de-tres 0 0 1)
0
>
Determine language from source custom
18:2
500.03 MB


Untitled - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
Untitled (define ...) Debug Macro Stepper Run Stop
1 #lang racket
2
Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.
> (define (menor x y)
(if (< x y) x y))
> (menor 10 9)
9
> (menor 4 5)
4
> (menor 4 3.9)
3.9
> (menor 4.4 4)
4
> (menor 3 0)
0
> (menor 2 2)
2
>
Determine language from source custom
17:2
507.62 MB


Untitled - DrRacket
File Edit View Language Racket Insert Scripts Tabs Help
Untitled (define ...) Debug Macro Stepper Run Stop
1 #lang racket
2
Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.
> (define (menor x y)
(if (< x y) x y))
> (define (menor-de-tres-v2 n1 n2 n3)
(menor (menor n1 n2) n3)
)
> (menor-de-tres-v2 2 8 1)
1
> (menor-de-tres-v2 3 0 3)
0
> (menor-de-tres-v2 3 3 3)
3
> (menor-de-tres-v2 0 1 0)
0
> (menor-de-tres-v2 1 0 0)
0
> (menor-de-tres-v2 1 10 9)
1
>
Determine language from source custom
20:2
507.72 MB

Ejercicio 8.

a) Suponiendo las definiciones:

```
(define (f x y)
    (cons x y))
(define (g x)
    (cons 2 x))
```

Realizar la evaluación paso a paso de la siguiente expresión:

(f (g (+ 2 1)) (+ 1 1))

▶ **Orden Aplicativo**

(f (g (+ 2 1)) (+ 1 1)) → ; Evaluamos (+ 2 1).

(f (g 3) (+ 1 1)) → ; Sustituimos g por su cuerpo.

(f (cons 2 3) (+ 1 1)) → ; Evaluamos (cons 2 3).

(f (2 . 3) (+ 1 1)) → ; Evaluamos (+ 1 1).

(f (2 . 3) 2) → ; Sustituimos f por su cuerpo.

(cons (2 . 3) 2) → ; Evaluamos (cons (2 . 3) 2).

((2 . 3) . 2) ←

▶ **Orden Normal**

(f (g (+ 2 1)) (+ 1 1)) → ; Sustituimos f por su cuerpo.

(cons (g (+ 2 1)) (+ 1 1)) → ; Sustituimos g por su cuerpo.

(cons (cons 2 (+ 2 1)) (+ 1 1)) → ; Evaluamos (+ 2 1).

(cons (cons 2 3) (+ 1 1)) → ; Evaluamos (+ 1 1).

(cons (cons 2 3) 2) → ; Evaluamos (cons 2 3).

(cons (2 . 3) 2) → ; Evaluamos (cons (2 . 3) 2).

((2 . 3) . 2) ←

b) Suponiendo las definiciones:

```
(define (f x)
        (/ x 0))
(define (g x y)
        (if (= x 0)
                0
                y))
```

Realizar la evaluación paso a paso de la siguiente expresión:

(g 0 (f 10))

▶ Orden Aplicativo

(g 0 (f 10)) ⟶ ¡Sustituimos f por su cuerpo.

(g 0 (/ 10 0)) ⟶ ¡Evaluamos (/ 10 0).

> Error: (División entre cero).

▶ Orden Normal

(g 0 (f 10)) ⟶ ¡Sustituimos g por su cuerpo.

(if (= 0 0) 0 (f 10)) ⟶ ¡Sustituimos f por su cuerpo.

(if (= 0 0) 0 (/ 10 0)) ⟶ ¡Evaluamos (= 0 0).

(if #t 0 (/ 10 0)) ⟶ ¡Evaluamos (/ 10 0).

> Error: (División entre 0).

Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.

```
> ;Función auxiliar del material
(define (añade-al-final x lista)
(append lista (list x)))
```

Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.

```
> (define(cadenas-mayores listal lista2)
    (añade-al-final (if (< (string-length(first (rest (rest listal)))) (string-length(first (rest (rest lista2))))) (first (rest (rest lista2))) (first (rest (rest listal))))
                    (añade-al-final (if (< (string-length(first (rest listal))) (string-length(first (rest lista2)))) (first (rest lista2)) (first (rest listal)))
                                    (cons (if (< (string-length(first listal)) (string-length(first lista2))) (first lista2) (first listal)) '()))))
```

Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.

Función auxiliar del material.

```
> (define (añade-al-final x lista)
(append lista (list x)))
> (define(cadenas-mayores listal lista2)
    (añade-al-final (if (< (string-length(first (rest (rest listal)))) (string-length(first (rest (rest lista2))))) (first (rest (rest lista2))) (first (rest (rest listal))))
                    (añade-al-final (if (< (string-length(first (rest listal))) (string-length(first (rest lista2)))) (first (rest lista2)) (first (rest listal)))
                                    (cons (if (< (string-length(first listal)) (string-length(first lista2))) (first lista2) (first listal)) '()))))

> (cadenas-mayores '("hola" "que" "tal") '("meme" "y" "adios"))
("hola" "que" "adios")
> (cadenas-mayores '("esto" "es" "lpp") '("hoy" "hay" "clase"))
("esto" "hay" "clase")
>
```

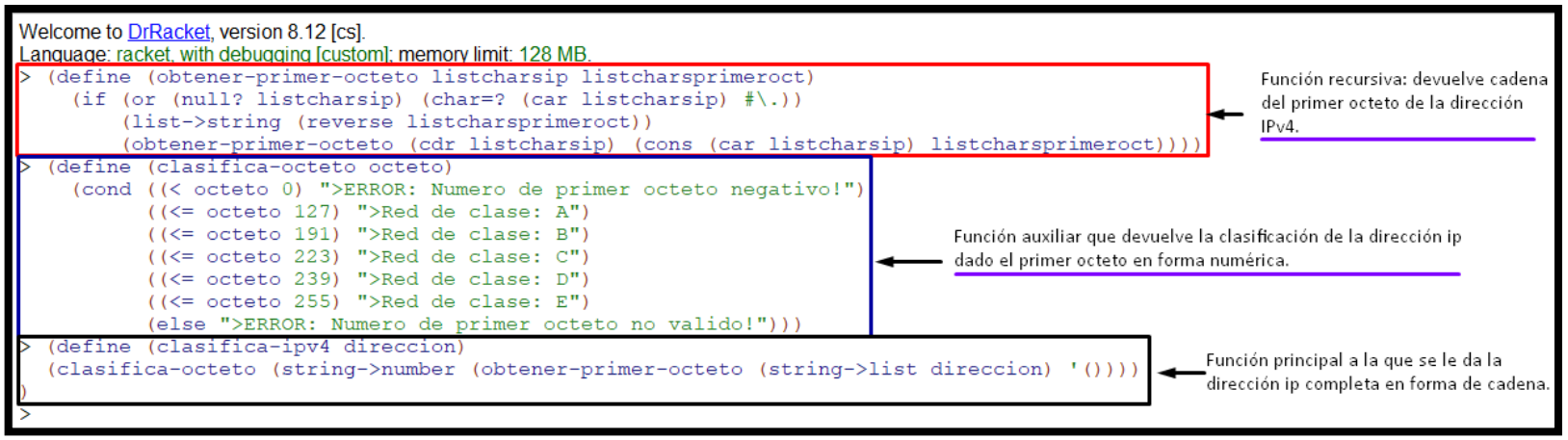
```
Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.
> (define (obten-palo charpalo)
    (cond
      ((= (char->integer charpalo) 66) 'Bastos)
      ((= (char->integer charpalo) 67) 'Copas)
      ((= (char->integer charpalo) 69) 'Espadas)
      ((= (char->integer charpalo) 79) 'Oros)
      (else "Caracter de palo invalido!")))
> (obten-palo #\O)
Oros
> (obten-palo #\E)
Espadas
> (obten-palo #\C)
Copas
> (obten-palo #\B)
Bastos
> (obten-palo #\D)
"Caracter de palo invalido!"
>
```

```
Welcome to DrRacket, version 8.12 [cs].
Language: racket, with debugging [custom]; memory limit: 128 MB.
> (define (obten-valor charvalor)
    (cond
      ((= (char->integer charvalor) 65) 1)
      ((= (char->integer charvalor) 83) 10)
      ((= (char->integer charvalor) 67) 11)
      ((= (char->integer charvalor) 82) 12)
      ((= (char->integer charvalor) 50) 2)
      ((= (char->integer charvalor) 51) 3)
      ((= (char->integer charvalor) 52) 4)
      ((= (char->integer charvalor) 53) 5)
      ((= (char->integer charvalor) 54) 6)
      ((= (char->integer charvalor) 55) 7)
      (else "Caracter de valor invalido!")
     )
    )
> (obten-valor #\A)
1
> (obten-valor #\S)
10
> (obten-valor #\C)
11
> (obten-valor #\R)
12
```

```
    )
> (obten-valor #\A)
1
> (obten-valor #\S)
10
> (obten-valor #\C)
11
> (obten-valor #\R)
12
> (obten-valor #\2)
2
> (obten-valor #\3)
3
> (obten-valor #\4)
4
> (obten-valor #\5)
5
> (obten-valor #\6)
6
> (obten-valor #\7)
7
> (obten-valor #\9)
"Caracter de valor invalido!"
>
```

```
> (define (carta simbolo)
    (cons (obten-valor (string-ref (symbol->string simbolo) 0)) (obten-palo (string-ref (symbol->string simbolo) 1))))
> (define tres-de-oros '3O)
> (define as-de-copas 'AC)
> (define caballo-de-espadas 'CE)
> (carta tres-de-oros)
(3 . Oros)
> (carta as-de-copas)
(1 . Copas)
> (carta caballo-de-espadas)
(11 . Espadas)
> (carta 'RB)
(12 . Bastos)
>
```

>Programa en Scheme que clasifique una dirección IP en función del primer octeto evitando el uso explícito de ciclos, utilizando en su lugar funciones recursivas puras y parámetros adicionales.




```
> (define (clasifica-ipv4 direccion)
  (clasifica-octeto (string->number (obtener-primer-octeto (string->list direccion) '()))))
)
> (clasifica-ipv4 "10.10.20.1")
">Red de clase: A"
> (clasifica-ipv4 "128.90.1.10")
">Red de clase: B"
> (clasifica-ipv4 "192.168.10.1")
">Red de clase: C"
> (clasifica-ipv4 "224.0.0.0")
">Red de clase: D"
> (clasifica-ipv4 "255.255.255.255")
">Red de clase: E"
>
```

Explicación de la función recursiva: Función llamada 'obtener-primer-octeto', a la cual se le da como argumentos la lista de caracteres de la cadena de la dirección IP (listcharsip) y una lista que va "almacenando" los caracteres del primer octeto en orden inverso (listcharsprimeroct). Para establecer un caso base y un caso general se hace uso de la instrucción 'if', esta realiza la comprobación: Si 'listcharsip' está null o si el primer elemento de dicha lista de caracteres es igual a #\., invierte el orden de la lista 'listcharsprimeroct', la convierte a string y la devuelve como resultado… De lo contrario, continúa procesando la cadena llamándose a sí misma nuevamente, y se le dan ahora como argumentos el resto de la lista de caracteres de la cadena (cdr listcharsip) y 'listcharsprimeroct' agregando el carácter actual (car listcharsip).