1)

```
let x = 4 in
    let f = fun (y) x+y end in
        let x=5 in
            (f)(10)
        end
    end
end
```

↓

```
let f = fun (y) 4 + y end in
    let x=5 in
        (f) (10)
    end
end
```

↓

```
let x=5 in
    (fun (y) 4 + y end) (10)
```

end

$\downarrow$

$(\text{fn } (y)\ 4+y \text{ end})(10)$

$\downarrow$

$4+10$

$\downarrow$

14 //

2)

c)

```
case And(e1, e2) =>
    eval(e1) match {
        case TrueV() => eval(e2)
        case FalseV() => FalseV()
    }

case Or(e1, e2) =>
    eval(e1) match {
        case TrueV() => TrueV()
        case FalseV() => eval(e2)
    }

case Not(e) =>
    eval(e) match {
```

```
case True V(F)   False V()
case False V() =) True V()
)
```

b)

```
case And( False(), _ )  =>  False()
case And( True(), True() ) => True()
case And( True(), False() ) => False()   ]  ou  case And(True(), e) =>e
case And( true(), e )  =>  And( True(), step(e) ) _]

case And(e1, e2) => And( step(e1), e2)

case Or( True(), _ )  =>  True()

case Or( False(), true()) => True()
case Or( False(), False() )=> False()    ]  case Or(False(), e) =>e
case Or( False(), e )  => Or( False, step(e)) _]

case Or(e1, e2) => Or( step(e1), e2)

case Not( True() ) => False()

case Not( False()) => True()

case Not(e)  => Not(step(e))
```

3)

```
params :: List(String)     Ex: a, b, c

corpo :: Exp              Ex: a + b + c
```

\   /

term(e)   a, b   ?

fun (c)   c, b   params

fun (b)

fun (c)
c + b + c
end

end

end

→ corpo
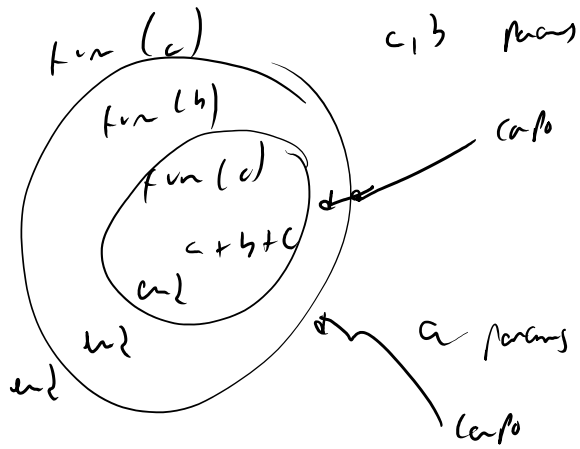
→ a params

→ corpo

Resposta:

yield params match {
        case Nil ⇒ Fun ("#", corpo)

        case _ ⇒ params.foldRight(corpo) (
                            (param, corpo) ⇒ Fun(param, corpo)
)
}