Parcial 3 Lenguaje de Programación:

Estudiante: Jonathan Bautista 16-10109

Pregunta 1:

El lenguaje escogido es Kotlin.

a.i) En kotlin es posible la definición(creación) de clases clases con constructores, campos, métodos, visibilidad e incluso tipos genéricos.

Ejemplo de esto:

```
abstract class Producto(val nombre : String){
   abstract  fun preguntar()
}

interface ComprarProducto{
   fun tomar_producto()
       fun pagar_producto(cantidad: Int ) = println("muchas gracias aquí tiene $cantidad dolares")
}

class SuperMercado(val nombre_producto: String ) : Producto(nombre_producto), ComprarProducto {
   fun tomar_carrito()= println("*agarrar carrito de compra*")
   override fun  tomar_producto() = println("*agarrar $nombre_producto*")
          override fun preguntar() = println("Muy buenas donde se encuentran el $nombre_producto")
}

fun main(){
   val sopa_instantanea= SuperMercado("Maruchan") // constructor
   sopa_instantanea.preguntar()// implementa una clase abstracta
   sopa_instantanea.tomar_carrito() // una funcion miembro
   sopa_instantanea.tomar_producto() // Implementa una interfaz
   sopa_instantanea.pagar_producto(10) // un metodo de una interfaz
}
```

Constructores:
En kotlin existen dos tipos de constructores.

El constructor primario o principal: hace parte de la cabecera de la clase. Este recibe como argumentos, aquellos datos que necesitas explicitamente para inicializar las propiedades al crear el objeto.

Constructores Secundarios: Este se usa en el caso de que los argumentos del constructor primario no satisfacen la creación de tu objeto, entonces puede crear un constructor secundario donde su declaración se realiza a través de constructor al interior de la clase.Si se tiene un constructor primario es obligatorio usar la expresión this para delegarle los parámetros que requiera(como en java).

Ejemplo:

Se requiere crear una clase producto donde uuid refleja el id del producto(código de barra) y nombre .
En este caso o se crea producto con el nombre y el uuid es generado por el programa
o se crea pasándole como argumento ambos.

```
class Producto(var nombre: String){
	var uuid: String

	init{
		uuid= UUID.randomUUID().toString()
	}
	constructor(uuid: String, nombre : String) : this(nombre){
		this.uuid= uuid
	}
}
```

```
Producto(“Arroz”)
Producto(“7857464”, ”Pasta”)
```

Como se mencionó anteriormente
Se declaró un constructor primario con el nombre y uuId generados automáticamente Luego en el constructor secundario se reciben ambos parámetros. Se delega hacia el constructor primario el nombre y el uuid se reemplaza.

Campos (propiedades):
En Kotlin, no existe el concepto de campo tal como se conoce; en su lugar, emplea el concepto de "propiedades".

Ejemplo, propiedades mutables (lectura-escritura) declaradas con la palabra reservada var: nombre en la clase Producto.

Visibilidad:
-public : Accesible en cualquier parte.
- private :Accesible sólo en el alcance actual
. Para clases se refiere a la clase que se está teniendo.
. Para declaraciones top–level se re ere al archivo actual
- protected: Accesible sólo en la clase actual y demás clases que hereden de ésta.
- internal : Accesible sólo en el módulo actual Nota: Si anula (override) un miembro protegido (protected) en la clase derivada sin especificar su visibilidad, su visibilidad también estará protegida (protected).

Si anula (override) un miembro protegido (protected) en la clase derivada sin especificar su visibilidad, su visibilidad también estará protegida (protected).

Ejemplo:

```
open class Padre() {
    var a = 1 // publica por defecto
    private var b = 2 // privado a clase Padre
    protected open val c = 3 // visible para la clase Padre y la clase hijo
    internal val d = 4 // visible dentro del mismo módulo
    protected fun e() { } // funcion visible para la clase Padre y la clase hijo
}
class Hijo: Padre() {
    // a, c, d y e() de la clase padre son visibles
// b no es visible
    override val c = 9 // c se enuentra protegido
}
fun main(args: Array<String>) {
    val padre = Padre()
// padre.a y padre.d son visibles
// padre.b, padre.c y padre.e() no son visibles
    val hijo = Hijo()

// hijo.c no es visible
}
```

Tipos Genericos: son tipos de parámetros especificados para las definiciones de clases, funciones e interfaces que escribe Kotlin en su código fuente.
Los tipos genéricos están marcados con <> después del nombre de la construcción.
Ejemplo:

```
class producto<T>
```

la T denota un tipo de marcador de posición, y es posible reemplazar la T con cualquier tipo de Kotlin cuando crea una instancia de un nuevo objeto de la clase, También es posible expresar más de un parámetro <T,U,V,...>

Dado que los parámetros de clase deben tener anotaciones de tipo, sólo es posible definir un tipo para que la clase lo acepte. Es un parámetro String o un parámetro Int.

Aquí es donde un tipo genérico puede ayudar. Al agregar un tipo de marcador de posición para la clase, se puede definir el tipo de los parámetros de la clase más adelante cuando se cree el objeto.

class producto(val nombre: T)
Ahora la T se puede reemplazar con un String o un Int según se requiera:

```
val producto  = producto("Arroz")
println(producto.nombre) // Arroz
 val otroproductor = producto(2)
 println(otroproducto.nombre) // 2
```

a.ii)

Crear Objetos:

En kotlin cuando se define la clase, solo se define la especificación para el objeto; no se asigna memoria ni almacenamiento. Para acceder a los miembros definidos dentro de la clase, es necesario crear objetos.

```
class Carro{
        private var velocidad : Int = 0

        fun aumentarVelocidad(val velocidad : Int ){
                this.velocidad=this.velocidad +velocidad
        }
        fun frenar(val velocidad : Int){
                this.velocidad= this.velocidad-velocidad
        }



}
fun main(args: Array) {
        val carro = Carro() // crear objeto carro de la clase Carro
        val carro2 = Carro() // crear objeto carro2 de la clase Carro

 }
```

Eliminar Objetos:

-Kotlin/JVM: No es necesario preocuparse por eliminar objetos como en c++/c... el recolector de basura de la JVM se encarga de ello, solo es necesario saber para eliminar un objeto es no mantener referencias en el objeto, si se tiene una colección (lista, mapa ...) donde se pone el objeto, también se debe sacarlo si la colección es una propiedad de una clase longeva como un modelo o del estilo , poner una referencia en una colección a la que hace referencia un objeto estático o de larga duración. Dentro de una función no es necesario eliminar los objetos creados con ella. Se debe tener en cuenta que el recolector de basura (GC) no se ejecuta instantáneamente después de finalizar el método.

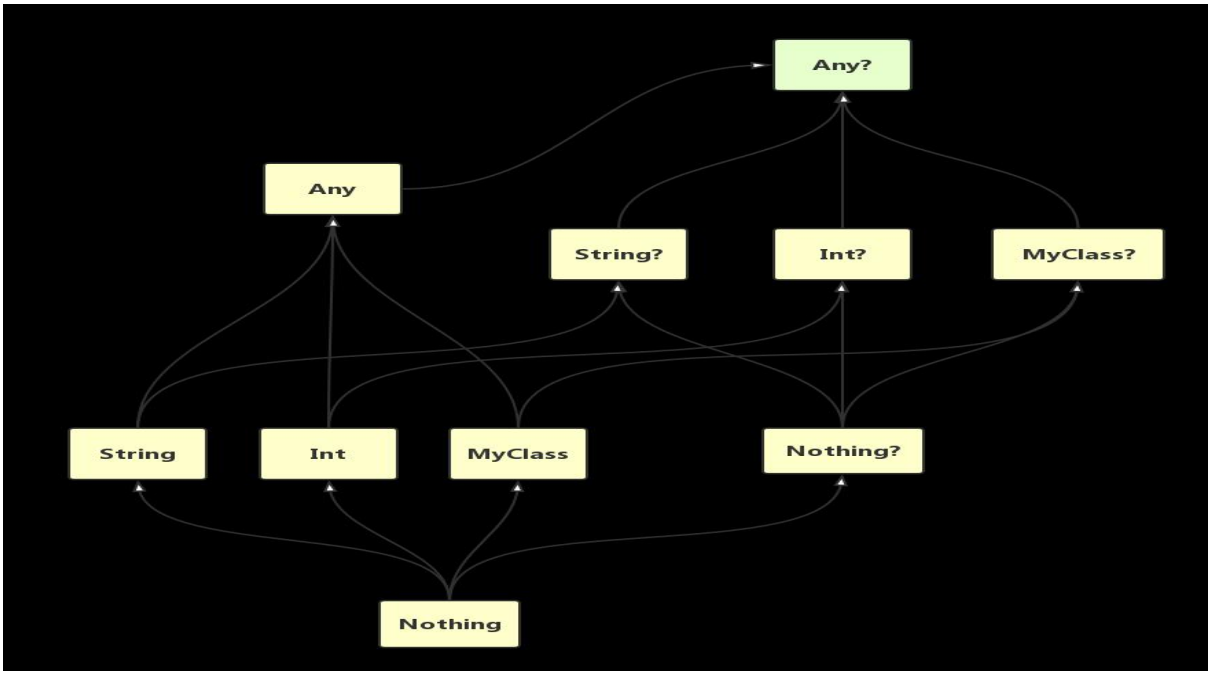
-Kotlin/Native: Kotlin / Native proporciona un esquema de administración de memoria automatizado, similar al que proporcionan Java o Swift. La implementación actual incluye un contador de referencia automatizado con un colector de ciclo para recoger basura cíclica.

iii)
El tipo de asociación que maneja es asociación estática de métodos y no es posible sobrecargar métodos ni propiedades.

Para que una clase admita sobrecargas en clases que lo heredan, la misma debe estar declarada como open. Los métodos y propiedades de igual forma.

En la clase hija deben establecer la sobrecarga explícitamente

iv)
En Kotlin, el tipo más alto en la jerarquía de tipos se llama Any. (Object en Java). Esto es decir todas las clases en Kotlin heredan explícitamente del tipo Any, incluyendo String, Int, Double, etc. El tipo Any contiene tres métodos: equals, toString, y hashcode.



Tipos primitivos: boolean, char, byte, short, int, long, float, double
Tipo de embalaje: booleano, carácter, byte, corto, entero, largo, flotante, doble.

Kotlin elimina el tipo original, solo "tipo envuelto". Cuando el compilador compila el código, optimiza automáticamente el rendimiento y desempaqueta el tipo de empaquetado correspondiente al tipo original. Los tipos de sistema Kotlin se dividen en tipos anulables y tipos no anulables. Kotlin introduce tipos anulables y usa tipos anulables para representar valores que pueden ser nulos. Esto establece un "límite" claro y explícito entre referencias anulables y no anulables.

Para los tipos de números de Kotlin, los tipos no anulables corresponden a los tipos de números primitivos en Java.

Kotlin	Java
Int	int
Long	long
Float	float
Double	double

El tipo de número anulable correspondiente en Kotlin es equivalente al tipo de número en caja en Java.

Kotlin	Java
Int?	Integer
Long?	Long
Float?	Float
Double?	Double

Kotlin posee clases abstractas:
-Estas son abiertas por defecto
-Los métodos que sobrecargan métodos y propiedades abstractas deben redefinirse con override
Kotlin tiene interfaces:
-Como la clases abstractas, pero no sin mantener estado
- Pueden proveer funcionalidad:
- definición de funciones por defecto
Kotlin no tiene herencia múltiple para clases:

- Para interfaces, sin embargo, si está disponible

```
interface A {
        fun f() {
                print("hola")
         }
}
interface B {
        fun f() {
                 print("chao")
         }
}
class C : A, B {
        override fun f() {
                 super<A>.f()
                 super<b>.f()
        }
}
```

- El polimorfismo en Kotlin es similar al de Java, osea polimorfismo general.

Manejo de varianza:
La varianza es la relación entre tipos. En general, el término varianza se refiere a la relación entre tipos genéricos que tienen la misma clase base pero diferentes argumentos de tipo. “La varianza establece la relación entre elementos genéricos que comparten la misma clase base” La varianza evita inconsistencias en los tipos de datos y evita bloqueos durante la ejecución del programa… tenemos 3 escenarios posibles dependiendo de la relación entre tipos genéricos:
-Invariancia: los componentes genéricos no tienen ninguna relación.
-Covarianza: el componente genérico con un parámetro de tipo derivado se considera hijo del componente con un parámetro de tipo base.
-Contravarianza: el componente genérico con un parámetro de tipo derivado se considera un padre del otro.

En Kotlin se tiene una forma de definir genéricos para que sean covariantes o contravariantes estos son in(contravariante) y Out(covariante).
Pregunta 1 b:

https://github.com/JKOMI2303/Lenguaje_Programacion-1/blob/main/Parcial%203/Pregunta_1/Pregunta_1_coleccion.kt

Pregunta 3:
X=1 Y=0 Z=9

Classes.txt

```
1   class Abra {
2       int a = 1, b = 0
3       fun cus(int x): int {
4           a = b + x
5           return pide(a)
6       }
7       fun pide(int y): int {
8           return a - y * b
9       }
10  }
11
12  class Cadabra extends Abra {
13      Abra zo = new PataDeCabra()
14      fun pide(int y): int {
15          return zo.cus(a + b) - y
16      }
17  }
18
19  class PataDeCabra extends Cadabra {
20      int b = 9, c = 9
21      fun cus(int x): int {
22          a = x - 3
23          c = a +b  * c
24          return pide(a * b + x)
25      }
26      fun pide(int y): int {
27          return c - y * a
28      }
29  }
30
31  Abra ho = new Cadabra()
32  Abra po = new PataDeCabra()
33  Cadabra cir = new PataDeCabra()
34  print(ho.cus(1) + po.cus(1) + cir.cus(1))
```

Asociacion Estatica de Metodos

Objeto	ho (Cadabra())	Po (Patadecabra())	cir (Patadecabra())	ho.zo (Patadecabra())	Pozo (Patadecabra())	Cr.zo (Patadecabra())
Metodos	cws -> Abra::cws pide -> Abra::pide	cws -> Abra::cws pide -> Abra::pide	cws -> Abra::cws pide -> Abra::pide	cws -> Abra::cws pide -> Abra::pide	cws -> Abra::cws pide -> Abra::pide	cws -> Abra::cws pide -> Abra::pide
Variable	a = 1 b = 0	a = 1 c = 9 b = 9	a = 1 c = 9 b = 4	a = 1 c = 1 b = 4	a = 1 c = 1 b = 9	a = 1 b = 4 c = 2

Imprime
0 + (-80) + (-80) = -160

ho.pide

Return	1-2*0	
y	1	
Pc	8	

ho.cws

Return	-	Pide(1)=0
a	-	0+1
x	1	1
Pc	4	5

Global Pc 34

Po cws

Return	10+10*9
y	10
Pc	8

Po.cws

Return	-	Pide(10) = 80
a	-	9+1
x	1	1
Pc	4	5

global Pc 34

Cir.pide

Return	10-10*9
y	10
Pc	8

Cir.cws

Return	-	Pide(10)
a	-	9+1
x	1	1
Pc	4	5

Global Pc 34

Asociacion Dinamica de Metodos

Objeto	ho (Cadabra())	Po (Patadecabra())	Cir (Patadecabra())	ho.zo (Patadecabra())	Po.zo (Patadecabra())	Cr.zo (Patadecabra())
Metodos	cws -> Abra::cws pide -> Cadabra::pide	cws -> Patadecabra::cws pide -> Patadecabra::pide	cws -> Patadecabra::cws pide -> Patadecabra::pide	cws -> Patadecabra::cws pide -> Patadecabra::pide	cws -> Patadecabra::cws pide -> Patadecabra::pide	cws -> Patadecabra::cws pide -> Patadecabra::pide
Variable	a = 1 b = 0	a = 1 c = 9 b = 1	a = 1 c = 4 b = 1	a = 1 c = 9 b = 9	a = 1 c = 4 b = 4	a = 1 c = 4 b = 9

Imprime
44 + 45 + 45 = 134.

zo.pide

Return	79-(-17)*(-2) = 45		
y	-17		
Pc	27		

zo.cws

Return	Pide(-17) = 45		
c	-	-	79
a	-	-2	
x	1	1	
Pc	22	23	24

ho.pide

Return	2ow(10)-1 = 44
y	1
Pc	15

ho.cws

Return	-	Pide(1) = 44
a	-	1
x	1	1
Pc	4	5

Global Pc 34

Po.pide

Return	79-(-17)*(-2) = 45
y	-17
Pc	27

Po.cws

Return	Pide(-17)		
c	-	-	79
a	-2		
x	1		
Pc	22	23	24

Global Pc 34

Cr.pide

Return	79-(-17)*(-2) = 45
y	-17
Pc	27

Cr.cws

Return	-	-	Pide(-17)
c	-	79	
a	-2		
x	1		
Pc	22	23	24

Global Pc 34

https://github.com/JKOMI2303/Lenguaje_Programacion-1/blob/main/Parcial%203/Pregunta_3/Classes.txt

Pregunta 4:
https://github.com/JKOMI2303/Lenguaje_Programacion-1/tree/main/Parcial%203/Pregunta_4

Pregunta 5:

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr _ e []        = e
foldr f e (x:xs)    = f x $ foldr f e xs

const :: a -> b -> a
const x _ = x

what :: a -> ([b] -> [(a, b)]) -> [b] -> [(a, b)]
what _ _ []     = []
what x f (y:ys) = (x, y) : f ys

--a)
misteriosa :: ???
misteriosa = foldr what (const [])

gen :: Int -> [Int]
gen n = n : gen (n + 1)

--a.i) orden de evaluacion normal Respuesta
 misteriosa "abc" (gen 1)
=
foldr what (const []) "abc" (gen 1)
=
what "a" $ foldr what (const []) "bc" (gen 1)
=
what "a" $ foldr what (const []) "bc" (1 : gen 2)
=
("a", 1) : $ foldr what (const []) "bc" (gen 2)
=
("a", 1) : what "b" $ foldr what (const []) "c" (gen 2)
=
("a", 1) : what "b" $ foldr what (const []) "c" (2 : gen 3)
=
("a", 1) : ("b", 2) : $ foldr what (const []) "c" (gen 3)
=
("a", 1) : ("b", 2) : what "c" $ foldr what (const []) "" (gen 3)
=
("a", 1) : ("b", 2) : what "c" $ foldr what (const []) "" (3 : gen 4)
```

```
=
("a", 1) : ("b", 2) : ("c", 3) $ foldr what (const []) "" (gen 4)
=
("a", 1) : ("b", 2) : ("c", 3) : (const []) (gen 4)
=
("a", 1) : ("b", 2) : ("c", 3) : []
=
("a", 1) : ("b", 2) : ("c", 3) : []
=
("a", 1) : ("b", 2) : [("c", 3)]
=
("a", 1) : [("b", 2), ("c", 3)]
=
[("a", 1), ("b", 2), ("c", 3)]

-- ii) Orde de evaluacion aplicativo Respuesta
misteriosa "abc" (gen 1)
=
misteriosa "abc" (1 : gen 2)
=
misteriosa "abc" (1 : 2 : gen 3)
=
misteriosa "abc" (1 : 2 : 3 : gen 4)
=
misteriosa "abc" (1 : 2 : 3 : 4 : gen 5)
=
misteriosa "abc" (1 : 2 : 3 : 4 : 6 : gen 6)
.
.
.
--Evaluacion recursiva infinita de gen

--5 b)
data Arbol a = Hoja | Rama a (Arbol a) (Arbol a)

--Su función debe cumplir con la siguiente firma:
foldA :: (a -> b -> b -> b) -> b -> Arbol a -> b

--5 b) Respuesta:

foldA :: (a -> b -> b -> b) -> b -> Arbol a -> b
foldA _ i Hoja = i
foldA f i (Rama value left right) = f value (foldA f i left) (foldA f i right)



--5 c)

whatTF :: a
```

```
   -> (Arbol b -> Arbol (a, b))
   -> (Arbol b -> Arbol (a, b))
   -> Arbol b
   -> Arbol (a, b)
whatTF _ _ _  Hoja          = Hoja
whatTF x f g (Rama y i d)   = Rama (x, y) (f i) (g d)

sospechosa :: ???
sospechosa = foldA whatTF (const Hoja)

genA :: Int -> Arbol Int
genA n = Rama n (genA (n + 1)) (genA (n * 2))

arbolito :: Arbol Char
arbolito = Rama 'a' (Rama 'b' Hoja (Rama 'c' Hoja Hoja)) Hoja

-- i) Orden de evaluacion Normal Respuesta:

sospechosa arbolito (genA 1)
=
foldA whatTF (const Hoja) arbolito (genA 1)
=
foldA whatTF (const Hoja) (Rama 'a'
                           (Rama 'b'
                                Hoja
                                (Rama 'c' Hoja Hoja)
                           )
                           Hoja
                     ) (genA 1)
=
whatTF 'a' (foldA whatTF (const Hoja) (Rama 'b'
                                       Hoja
                                       (Rama 'c'
                                            Hoja
                                            Hoja
                                       )
                                  )
           )
           (foldA whatTF (const Hoja) Hoja)
           (genA 1)
= -- Evaluamos (genA 1)
whatTF 'a' (foldA whatTF (const Hoja) (Rama 'b'
                                       Hoja
                                       (Rama 'c' Hoja Hoja)
                                  )
           (foldA whatTF (const Hoja) Hoja)
           (Rama 1
               (genA (1 + 1))
```

```
            (genA (1 * 2))
        )
= -- Evaluamos whatTF 'a' f g (Rama y i d) -:

--Rama ( x , y)

Rama ('a', 1)
    ((foldA whatTF (const Hoja) (Rama 'b'
                                Hoja
                                (Rama 'c' Hoja Hoja)
                            )
     )
     (genA (1 + 1))
    )
    ((foldA whatTF (const Hoja) Hoja)
     (genA (1 * 2))
    )
= -- Evaluamos el foldA que se encuentra más arriba
Rama ('a', 1)
     (whatTF 'b' (foldA whatTF (const Hoja) Hoja) (foldA whatTF (const Hoja) (Rama 'c' Hoja
Hoja)) (genA (1 + 1)))
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))
= -- Evaluamos (genA (1+1))

Rama ('a', 1)
     (whatTF 'b' (foldA whatTF (const Hoja) Hoja) (foldA whatTF (const Hoja) (Rama 'c' Hoja
Hoja)) (Rama 2
                                                                    (genA (2 + 1))    ----i
                                                                    (genA (2 * 2))    ----d
                                                                )
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))
=
-- Evaluamos whatTF 'b' ---:
Rama ('a', 1)
    (Rama ('b', 2)
        ((foldA whatTF (const Hoja) Hoja)              (genA (2 + 1)))
        (foldA whatTF (const Hoja) (Rama 'c' Hoja Hoja)  (genA (2 * 2)))
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos el foldA que se encuentra más arriba

Rama ('a', 1)
    (Rama ('b', 2)
        ((const Hoja) (genA (2 + 1)))
        (foldA whatTF (const Hoja) (Rama 'c' Hoja Hoja)  (genA (2 * 2)))
    )
```

```
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos (const Hoja) (genA (2 + 1))

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        (foldA whatTF (const Hoja) (Rama 'c' Hoja Hoja)  (genA (2 * 2)))
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos foldA de más arriba

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
          (whatTF 'c' (foldA whatTF (const Hoja) Hoja) (foldA whatTF (const Hoja) Hoja) (genA
(2 * 2)))
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos (genA (2 * 2))

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
         (whatTF 'c' (foldA whatTF (const Hoja) Hoja) (foldA whatTF (const Hoja) Hoja) (Rama
4
                                                                    (genA(4+1))
                                                                    (genA(4*2))
                                                                )
        )
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos whatTF 'c' ....

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        Rama ('c', 4)
            ((foldA whatTF (const Hoja) Hoja) (genA(4+1)))
            ((foldA whatTF (const Hoja) Hoja) (genA(4*2)))
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos foldA de más arriba
```

```
Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        Rama ('c', 4)
            ((const Hoja) (genA(4+1)))
            ((foldA whatTF (const Hoja) Hoja) (genA(4*2)))
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= --- Evaluamos (const Hoja) (genA(4+1))

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        Rama ('c', 4)
            Hoja
            ((foldA whatTF (const Hoja) Hoja) (genA(4*2)))
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos foldA de más arriba

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        Rama ('c', 4)
            Hoja
            ((const Hoja) (genA(4*2)))
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= --- Evaluamos (const Hoja) (genA(4*2))

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        Rama ('c', 4)
            Hoja
            Hoja
    )
    ((foldA whatTF (const Hoja) Hoja) (genA (1 * 2)))

= -- Evaluamos foldA de más arriba

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        Rama ('c', 4)
```

```
            Hoja
            Hoja
    )
    ((const Hoja) (genA (1 * 2)))

= -- Evaluamos (const Hoja) (genA (1 * 2))

Rama ('a', 1)
    (Rama ('b', 2)
        Hoja
        Rama ('c', 4)
            Hoja
            Hoja
    )
    Hoja

--- ii) Orden de evaluación Aplicativo Respuesta:


sospechosa arbolito (genA 1)
=
sospechosa arbolito Rama 1
                    (genA (2))
                    (genA (2))
=
sospechosa arbolito Rama 1
                    (Rama 2 (genA (3)) (genA (4)))
                    (genA (2))
=
sospechosa arbolito Rama 1
                    (Rama 2 (genA (3)) (genA (4)))
                    (Rama 2 (genA (3)) (genA (4)))
=
sospechosa arbolito Rama 1
                    (Rama 2
                        (genA (3))
                        (genA (4))
                    )
                    (Rama 2
                        (genA (3))
                        (genA (4))
                    )
=
sospechosa arbolito Rama 1
                    (Rama 2
                        (Rama 3 (genA (4)) (genA (6)))
                        (genA (4))
                    )
```

(Rama 2
(genA (3))
(genA (4))
)
=
sospechosa arbolito Rama 1
(Rama 2
(Rama 3 (genA (4)) (genA (6)))
(Rama 3 (genA (4)) (genA (6)))
)
(Rama 2
(genA (3))
(genA (4))
)
.
.
.
--Evaluacion recursiva infinita de genA.

https://github.com/JKOMI2303/Lenguaje_Programacion-1/blob/main/Parcial%203/Pregunta_5/main.hs