Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI3641 – Lenguajes de Programación 1

Trimestre: Septiembre - Diciembre 2023

Profesor: Ricardo Monascal

Estudiante: Junior Miguel Lara Torres, Carnet: 17-10303

Parcial 3 (30 pts)

- "En algunas preguntas, se usarán las constantes X, Y y Z. Estas constantes debe obtenerlas de los últimos tres números de su carnet."

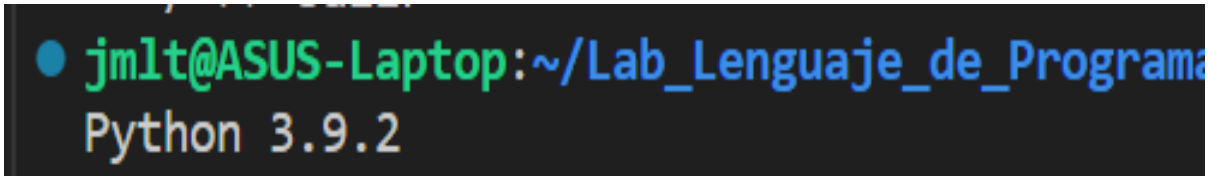
Caso particular, 17-10303 entonces $X = 3,\ \ Y = 0,\ \ Z = 3$.

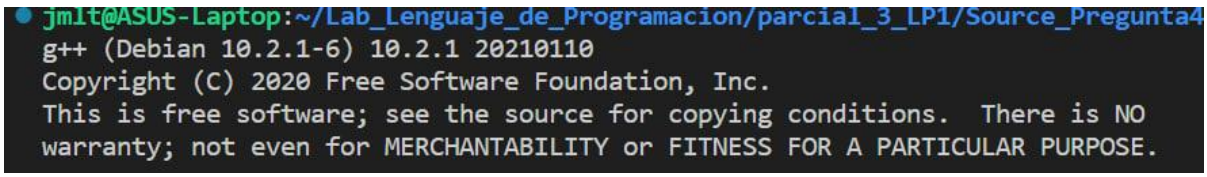
- "En aquellas preguntas donde se le pida implementar un programa, mantenga su código en un repositorio git remoto (preferiblemente Github) y coloque un enlace al mismo en lugar de su respuesta. Todo su código debe ser legible y estar debidamente documentado."

Todos los códigos, este propio documento, documento oficial del enunciado de examen 2 y otras evaluaciones de la materia serán cargados en:

https://github.com/JMLTUnderCode/Programming/tree/main/USB_Language_Programation

Para responder varias de las preguntas se usó Python3 y C++ cuyas versiones son:





- **1er Pregunta:**

El lenguaje utilizado en esta pregunta es C++, dado que mi nombre es Junior Miguel Lara Torres.

  - (a)
    - i.

C++ permite crear y manipular objetos a través de clases y structs. Los objetos se crean mediante constructores, que son métodos especiales para inicializar un objeto. Los objetos tienen campos de datos (variables miembro) y métodos (funciones miembro) que operan sobre esos datos.

```
class Profe {
  public:
    Persona(string nombre);
    ~Persona();

  private:
    string nombre;
}

Persona* p = new Profe("Ricardish");
// crea objeto Persona
delete p; // destruye el objeto
```

    - ii.

Este lenguaje maneja la memoria de forma explícita a través de los operadores ***new*** y ***delete***. El programador debe reservar y liberar memoria manualmente. No hay recolección de basura automática. Sin embargo, existe una librería de nombre ***Boehm-Demers-Weiser (BDW)*** que funciona como recolector de basura, es una biblioteca de C que puede ser utilizada en proyectos de C++ para gestionar la memoria dinámica.

    - iii.

En C++ se utiliza enlazado estático de métodos de forma predeterminada. Esto significa que los métodos se enlazan a la definición de la clase en tiempo de compilación. Pero se puede usar enlazado dinámico con punteros a funciones, la palabra clave que se usa para indicar que un método asocia dinámicamente es ***virtual***.

- iv.

  Es un lenguaje con tipado estático y fuerte. Soporta herencia simple y múltiple a través de clases base. Tiene polimorfismo paramétrico mediante ***template <typename T>***. Y maneja la varianza de tipos permitiendo especificar si una clase derivada es un subtipo covariante, contravariante o invariante de su clase base.

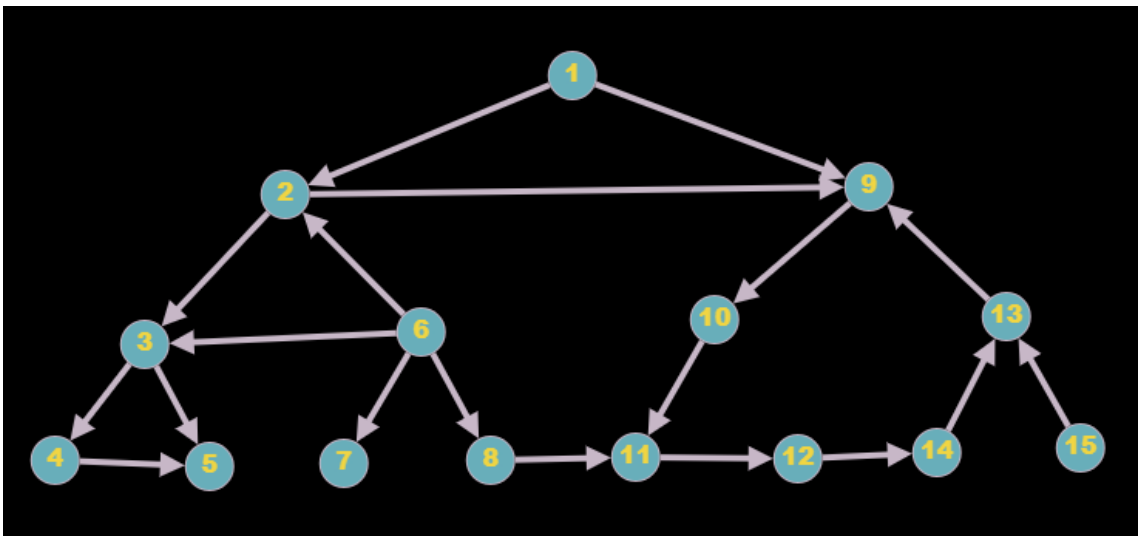
- (b)
  - i.

    El archivo principal de la respuesta a este apartado se encuentra en el siguiente link.

    https://github.com/JMLTUnderCode/Programming/blob/main/USB_Lenguage_Programation/Parcial_3/Source_Pregunta1/Pregunta_1_b_1_respuesta.cpp

  - ii.

    El grafo usado como ejemplo es el siguiente.



    El archivo principal de la respuesta a este apartado se encuentra en el siguiente link.

    https://github.com/JMLTUnderCode/Programming/blob/main/USB_Lenguage_Programation/Parcial_3/Source_Pregunta1/Pregunta_1_b_2_respuesta.cpp

- **2er Pregunta:**

El lenguaje escogido para esta pregunta es Python, pues mi nombre es Junior, 6 chars ambos.

a) Descripción de Python 3.

1) Python provee capacidades nativas limitadas para concurrencia a través de sus ***threads***. Sin embargo, debido a la Global Interpreter Lock ***(GIL)***, solo un ***thread*** puede ejecutarse a la vez, por lo que no permite verdadera ejecución concurrente en múltiples ***CPUs***. Para lograr concurrencia real, se deben usar librerías. Es decir, los ***threads*** de ***Python*** permiten concurrencia a nivel de flujo de ejecución, pero la existencia del ***GIL*** limita la verdadera ejecución concurrente en ***hardware multi-núcleo***. Para levantar esta limitación se requieren librerías externas como ***multiprocessing***.

2) La librería ***multiprocessing*** permite crear procesos como tareas concurrentes que tienen su propio espacio aislado de memoria. Esto evita el problema del ***GIL*** ya que los procesos se ejecutan realmente en paralelo.

Para crear un proceso se utiliza la clase ***Process***, pasándole como argumento la función que se desea ejecutar en paralelo:

```
from multiprocessing import Process

def funcion_paralela():
   # código a ejecutar en paralelo

proceso = Process(target=funcion_paralela)
proceso.start() # Inicia la ejecución en paralelo
```

También se pueden pasar argumentos a la función utilizando los argumentos ***args*** y ***kwargs*** de ***Process***.

La comunicación y sincronización entre procesos se logra a través de:

- Colas (***Queue***) para pasar mensajes de forma segura.
- ***Pipes*** para comunicación directa entre dos procesos.
- Memoria compartida (***Shared memory***) para compartir datos mutables.
- Primitivas de sincronización (***Locks, Events, Semaphores***, etc.)

El manejo de los procesos incluye poder terminarlos con ***.join()*** o **.terminate()**, comprobar si han finalizado, y obtener su valor de retorno una vez terminan.

3) Dado que los procesos en ***multiprocessing*** tienen su propio espacio aislado de memoria, se requiere sincronización para coordinar el acceso a recursos compartidos y evitar condiciones de carrera. Los principales mecanismos de sincronización son:
    - ***Locks***: Permite que solo un proceso acceda a un recurso compartido (o sección crítica) a la vez. Se adquiere el lock antes de acceder al recurso con ***.acquire()*** y se libera con ***.release().***
    - Semáforos (***Semaphore***): Similar a ***locks***, pero permite controlar cuantos procesos pueden acceder al recurso a la vez.
    - ***Events***: Permiten que procesos esperen hasta que un evento sea "seteado" para continuar la ejecución.
    - ***Conditions***: Permiten esperar hasta que se cumpla una condición arbitraria.
    - ***Barriers***: Obligan a que cierto número de procesos esperen en un punto hasta que todos lleguen.

    Estas primitivas se pasan a los procesos como argumentos y se utilizan para sincronizar el acceso y el orden de ejecución según sea necesario.

    Por ejemplo, se puede usar un ***Lock*** antes de acceder a una estructura de datos compartida, o un ***Event*** para coordinar que un proceso espere a otro.

b) Implementación de programas:

1) El archivo principal de la respuesta a este apartado se encuentra en el siguiente link.

2) El archivo principal de la respuesta a este apartado se encuentra en el siguiente link.

- **3er Pregunta:**

```
# Para X = 3, Y = 0, Z = 3
class Abra {
    int a = 3, b = 0
    fun cus(int x): int {
        a = b + x
        return pide(a)
    }
    fun pide(int y): int {
        return a - y * b
    }
}

class Cadabra extends Abra {
    Abra zo = new PataDeCabra()
    fun pide(int y): int {
        return zo.cus(a + b) - y
    }
}

class PataDeCabra extends Cadabra {
    int b = 3, c = 3
    fun cus(int x): int {
        a = x - 3
        c = a + b * c
        return pide(a * b + x)
    }
    fun pide(int y): int {
        return c - y * a
    }
}

Abra ho = new Cadabra()
Abra po = new PataDeCabra()
Cadabra cir = new PataDeCabra()
print(ho.cus(4) + po.cus(1) + cir.cus(4))
```

- ***Modo Estático***

    - La variable ***"ho"*** tiene las siguientes características.
        - Se le presta atención a su valor ***estático***, es decir ***Abra***.
        - Tiene como valores iniciales ***a = 3*** y ***b = 0*** que viene de ***Abra***.
        - Posee las funciones ***cus*** y ***pide*** definidas en ***Abra***.

    - La variable **"po"** tiene las siguientes características.
        - Se le presta atención a su valor ***estático***, es decir ***Abra***.
        - Tiene como valores iniciales ***a = 3*** y ***b = 0*** que viene de ***Abra***.
        - Posee las funciones ***cus*** y ***pide*** definidas en ***Abra***.

    - La variable **"cir"** tiene las siguientes características.
        - Se le presta atención a su valor estático, es decir ***Cadabra***.
        - Tiene como valores iniciales que vienen de ***Abra***, ***a = 3***, ***b = 0*** e internamente ***"zo"*** de tipo ***Abra***, por lo que a dicho ***"zo"*** se le presta atención a su valor estático, es decir ***Abra***.
        - Posee la función ***pide*** definida en ***Cadabra*** y ***cus*** heredada de ***Abra***.

    - Entonces, para las siguientes llamadas.
        - ho.cus(4) -> pide(0+4) -> (0+4) – (0+4)*0 -> **4**
        - po.cus(1) -> pide(0+1) -> (0+1) – (0+1)*0 -> **1**
        - cir.cus(4) ->pide(0+4) -> zo.cus(0+4+0) – (0+4) -> pide(0+3+0) – (0 + 4) -> (0+4) - (0+3+0)*0 - (0 + 4) -> **0**

    - Resultado final: print(4 + 1 + 0) -> **5**

- ***Modo Dinámico***

    - La variable ***"ho"*** tiene las siguientes características.
        - Se le presta atención a su valor ***dinámico***, es decir ***Cadabra***.
        - Tiene como valores iniciales ***a = 3*** y ***b = 0*** que viene de ***Abra*** e internamente ***"zo"*** al cual se le asocia dinámicamente con su valor dinámico ***PataDeCabra***.
        - Posee las funciones ***cus*** heredada de Abra y ***pide*** definida como en interna.

    - La variable **"po"** tiene las siguientes características.
        - Se le presta atención a su valor ***dinámico***, es decir ***PataDeCabra***.
        - Tiene como valores iniciales ***a = 3*** heredado de ***Abra***, luego ***b = 3*** y ***c = 3*** definidas en ***PataDeCabra***.

- Posee las funciones ***cus*** y ***pide*** definidas en ***PataDeCabra***.

- La variable **"cir"** tiene las siguientes características.
  - Se le presta atención a su valor ***dinámico***, es decir ***PataDeCabra***.
  - Tiene como valores iniciales ***a = 3*** heredado de ***Abra***, luego ***b = 3*** y ***c = 3*** definidas en ***PataDeCabra***.
  - Posee las funciones ***cus*** y ***pide*** definidas en ***PataDeCabra***.

- Entonces, para las siguientes llamadas.
  - ho.cus(4) ->
  - po.cus(1) ->
  - cir.cus(4) ->

- Resultado final: print() ->

- **4er Pregunta:**

  Se escoge Python3 como lenguaje de programación.

  En el siguiente link encontrara los siguientes archivos esenciales:
  ➔ Pregunta_4_respuesta.py
  ➔ virtualtable.py
  ➔ test_ virtualtable.py

  https://github.com/JMLTUnderCode/Programming/tree/main/USB_Lenguage_Programation/Parcial_3/Source_Pregunta4

  La ejecución del archivo fuente principal se realiza por consola con:

  ```
  [py o python3] Pregunta_2_respuesta.py
  ```

  En el caso de la cobertura se requiere que haga la instalación **`pytest`** y **`coverage`** en su sistema operativo basado en Linux. Es decir, realice los siguientes pasos:

  ```
  pip install pytest
  pip install coverage
  ```

  Luego debe realizar la siguiente corrida en consola:

  ```
  coverage run -m pytest test_AritmeticExprCalculator.py
  ```

  Mostrándole en pantalla la ejecución correcta de las pruebas unitarias. Luego para ver la información detallada de la cobertura debe escribir en consola:

  ```
  coverage report -m
  ```

  Cuyos resultados corresponden a un 96% de cobertura.

- **5er Pregunta:**
  - (a)

```
-- 1) Evaluacion normal

misteriosa "abc" (gen 1)

== -- Definicion de misteriosa

foldr what (const []) "abc" [1, 2, ...]

== -- Definicion de foldr

(a, 1) : foldr what (const []) "bc" [2, 3, ...]

== -- Definicion de foldr

(a, 1) : (b, 2) : foldr what (const []) "c" [3, 4, ...]

== -- Definicion de foldr

(a, 1) : (b, 2) : (c, 3) : foldr what (const []) "" [4, 5, ...]

== -- Definicion de foldr en caso base "foldr _ e [] = e"

(a, 1) : (b, 2) : (c, 3) : (const [])

== -- Definicion de const, "const x _ = x"

(a, 1) : (b, 2) : (c, 3) : []

== -- Concatenacion de elementos como una lista.

[(a, 1), (b, 2), (c, 3)]

-----------------------------------------------------------

-- 2) Evaluacion aplicativa

misteriosa "abc" (gen 1)

== -- Definicion de foldr

foldr what (const []) "abc" (gen 1)

== -- Definicion de foldr generando una lambda funcion para luego
ser aplicada a (gen 1)

(\xs -> foldr what (const []) "abc" xs) (gen 1)

== -- Desarrollando ahora (gen 1) y luego aplicando lambda.

foldr what (const []) "abc" [1,2,...]

== -- Definicion de foldr

(...) -- Mismos pasos que en la version de evaluacion normal.

==

(a, 1) : (b, 2) : (c, 3) : []

==

[(a, 1), (b, 2), (c, 3)]
```

- (b)

```haskell
data Arbol a = Hoja | Rama a (Arbol a) (Arbol a)

foldA :: (a -> b -> b -> b) -> b -> Arbol a -> b
foldA _ e Hoja = e
foldA f e (Rama x a b) = f x (foldA f e a) (foldA f e b)
```

- (c)

Completamos la firma de ***sospechosa*.**

```haskell
sospechosa :: Arbol a -> Arbol b -> Arbol (a, b)
sospechosa = foldA whatTF (const Hoja)
```

Primero tengamos en cuenta que ***(genA 1)*** genera un árbol infinito, tal como se observa en la imagen en la parte de descripción de ***(genA 1)***

Luego, observamos el desglose de arbolito en este mismo sentido. Ahora si queremos realizar "***sospechosa arbolito (genA 1)***" debemos considerar que en la versión de evaluación NORMAL tenemos como inicialmente (genA 1) se expande de tal forma que crea un árbol infinito, la imagen de la siguiente pagina explica el procedimiento paso a paso para recudir la expresión.

```
-- Descripcion de genA 1
Rama 1
    Rama 2
        genA 3
            (...)
        genA 4
            (...)

    Rama 2
        genA 3
            (...)
        genA 4
            (...)

-- Descripcion de arbolito
Rama 'a'
    Rama 'b'
        Hoja
        Rama 'c'
            Hoja
            Hoja

    Hoja
```

```
-- Descripcion de la version de evaluacion normal.

sospechosa arbolito (genA 1)

== -- Definicion de genA 1, desglosada uno 2 a 3 terminos.

sospechosa arbolito (Rama 1 (Rama 2 (genA 3) genA(4)) (Rama 2 (genA 3) (genA 4)))

== -- Definicion de arbolito

sospechosa
    (Rama 'a' (Rama 'b' Hoja (Rama 'c' Hoja Hoja)) Hoja)
    (Rama 1 (Rama 2 (genA 3) genA(4)) (Rama 2 (genA 3) (genA 4)))

== -- Definicion de sospechosa

foldA whatTF (const Hoja)
    (Rama 'a' (Rama 'b' Hoja (Rama 'c' Hoja Hoja)) Hoja)
    (Rama 1 (Rama 2 (genA 3) genA(4)) (Rama 2 (genA 3) (genA 4)))

== -- Definicion de foldA

whatTF 'a'
    ( foldA whatTF (const Hoja) (Rama 'b' Hoja (Rama 'c' Hoja Hoja)) )
    ( foldA whatTF (const Hoja) Hoja  )
    (Rama 1 (Rama 2 (genA 3) genA(4)) (Rama 2 (genA 3) (genA 4)))

== -- Definición de whatTF

Rama ('a', 1)
     ( foldA whatTF (const Hoja) (Rama 'b' Hoja (Rama 'c' Hoja Hoja)) ) ( Rama 2 (genA 3) (genA 4) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de foldA en subárbol izquierdo

Rama ('a', 1)
     ( whatTF 'b'
            (foldA whatTF (const Hoja) Hoja)
            (foldA whatTF (const Hoja) (Rama 'c' Hoja Hoja) )
            (Rama 2 (genA 3) (genA 4)) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de whatTF

Rama ('a', 1)
     ( Rama ('b', 2)
            ( foldA whatTF (const Hoja) Hoja (Rama 3 (genA 4) (genA 6)) )
            ( foldA whatTF (const Hoja) (Rama 'c' Hoja Hoja) ( Rama 4 (genA 5) (genA 8)) ) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de foldA para caso base en subárbol izquierdo

Rama ('a', 1)
     ( Rama ('b', 2)
            ( const Hoja (Rama 3 (genA 4) (genA 6)) )
            ( foldA whatTF (const Hoja) (Rama 'c' Hoja Hoja) ( Rama 4 (genA 5) (genA 8)) ) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de const

Rama ('a', 1)
     ( Rama ('b', 2)
            Hoja
            ( foldA whatTF (const Hoja) (Rama 'c' Hoja Hoja) ( Rama 4 (genA 5) (genA 8)) ) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )
```

```
== -- Definicion de foldA en subarbol derecho del subarbol izquierdo.

Rama ('a', 1)
     ( Rama ('b', 2)
            Hoja
            ( whatTF 'c'
                    ( foldA whatTF (const Hoja) Hoja )
                    ( foldA whatTF (const Hoja) Hoja )
                    ( Rama 4 (genA 5) (genA 8)) ) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definiciones de foldA en casos bases

Rama ('a', 1)
     ( Rama ('b', 2)
            Hoja
            ( whatTF 'c'
                    ( const Hoja )
                    ( const Hoja )
                    ( Rama 4 (genA 5) (genA 8)) ) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de whatTF

Rama ('a', 1)
     ( Rama ('b', 2)
            Hoja
            ( Rama ('c', 4)
                    ( const Hoja ) (genA 5)
                    ( const Hoja ) (genA 8) )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de const

Rama ('a', 1)
     ( Rama ('b', 2)
            Hoja
            ( Rama ('c', 4)
                    Hoja
                    Hoja )
     ( foldA whatTF (const Hoja) Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de foldA en caso base para sub arbol derecho.

Rama ('a', 1)
     ( Rama ('b', 2)
            Hoja
            ( Rama ('c', 4)
                    Hoja
                    Hoja )
     ( const Hoja (Rama 2 (genA 3) (genA 4)) )

== -- Definicion de const

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

== -- Expresion lineal

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

Ahora se muestra a continuación la versión de evaluación aplicativa. En este la diferencia es que ***(genA 1)*** no se genera antes de ser pasado como argumento al igual que ***arbolito***, por tanto, solo al final luego del desarrollo de aplicaciones parciales (lambdas) llegaremos a un desarrollo similar a la evaluación normal. Acá un desarrollo previo, pero en términos finales se tiene la misma expresión que en la evaluación normal. Es decir, la diferencia clave es que en la evaluación aplicativa los argumentos se mantienen como funciones sin evaluar el mayor tiempo posible. Pero al final, para poder construir el árbol resultado, se tienen que acabar evaluando completamente de forma equivalente a la normal.

```
-- Descripcion de la version de evaluacion aplicativa.

== -- Definicion de sospechosa como funcion lambda.

(\t -> foldA whatTF (const Hoja) t ) arbolito (genA 1)

== -- arbolito es un valor, se sustituye

(\t -> foldA whatTF (const Hoja) t )
    ( Rama 'a' (Rama 'b' Hoja (Rama 'c' Hoja Hoja)) Hoja) )
    ( genA 1 )

== -- genA 1 desarrollado parcialmente

(\t -> foldA whatTF (const Hoja) t )
    ( Rama 'a' (Rama 'b' Hoja (Rama 'c' Hoja Hoja)) Hoja) )
    (\n -> Rama n (genA (n+1)) (genA (n*2))) 1

== -- Definicion de foldA parcialmente

( (\x y z -> whatTF x (foldA whatTF (const Hoja) y) (foldA whatTF (const Hoja) z))
    'a'
    (Rama 'b' Hoja (Rama 'c' Hoja Hoja))
    Hoja )
    (\n -> Rama n (genA (n+1)) (genA (n*2))) 1

== -- foldA se aplica parcialmente al subárbol izquierdo

( (\x y z -> whatTF x
    ((\a b c -> whatTF a b c) whatTF (const Hoja) y)
    (foldA whatTF (const Hoja) z))
    'a'
    (Rama 'b' Hoja (Rama 'c' Hoja Hoja))
    Hoja )
    (\n -> Rama n (genA (n+1)) (genA (n*2))) 1

== (Continuacion...)
```

- **6er Pregunta:**

- **Pregunta Extra:**

Sin tiempo. ☹