

# PARADIGMAS DE PROGRAMACIÓN

## Parcial I - Resumen y Prácticas

### Temas:

- Programación declarativa (paradigma funcional)  
Diferencias c/ q. imperativo.
- Pasaje de parámetros: por valor; por referencia; por valor-resultado; por necesidad; por nombre.
- Alcance: dinámico y estático
- Excepciones
- Tipado: inferencia de tipos

### Índice:

|                            |                       |    |
|----------------------------|-----------------------|----|
| • Resumen                  | - - - - -             | 2  |
| - Programación declarativa | - - - - -             | 2  |
| - Pasaje de parámetros     | - - - - -             | 3  |
| - Alcance                  | - - - - - - - - -     | 5  |
| - Excepciones              | - - - - - - - - -     | 7  |
| - Tipado                   | - - - - - - - - -     | 9  |
| • Práctica                 | - - - - - - - - -     | 13 |
| - Ejercicios del libro     | - - - - - - - - - - - | 13 |
| - Recuperatorio 10/06/24   | - - - - -             | 18 |
| - Recuperatorio 11/06/24   | - - - - -             | 24 |
| - Parcial 09/05/23         | - - - - -             | 30 |

## Resumen:

### Programación declarativa

"cómo son las cosas"

Los lenguajes declarativos crean nuevos valores que son **inmutables**.

⇒ no existe la **asignación destructiva**

Ej:  $\text{int } x = 0$

$x = x + 1 \rightarrow$  en prog. declarativa

**NO** se quede

ya que tendría tener efectos secundarios en el resto del programa.

Una operación declarativa es:

→ **independiente** (del estado de la computación; depende sólo de sus argumentos)

→ **sin estado** (no recuerda estados entre llamados)

→ **determinística** (los llamados con los mismos argumentos dan siempre los mismos resultados)

*consecuencia*

**Transparencia referencial**

Una expresión se puede sustituir por su valor sin cambiar la semántica del programa.

↳ todas las componentes declarativas pueden usarse como valores.

OP. declarativa (log. declarativa) → OP. declarativa  
composición de operaciones declarativas es una operación declarativa.

# Pasaje de parámetros

**Argumento:** expresión que aparece cuando llamo a la función.

**Parámetro:** identificador que aparece en la declaración de la función.

fun A (x, y) : → parámetros: x, y

$x := x + 1$ ;  $y := y - 1$

$A(a, b)$  → argumentos: a, b

~~x := 8~~  
l-valor      r-valor

## Mecanismos de pasaje de parámetros

- **Pasaje por valor:** la función que llama pasa el r-valor del argumento.

ejemplo anterior:

$$a = 3; b = 4 \Rightarrow A(a, b) = A(3, 4)$$

la función no puede cambiar el valor de la variable de la función que llama a y b siempre valen 3 y 4.

- **Pasaje por referencia:** la función que llama pasa el l-valor del argumento.

ejemplo anterior:

$$a = \overset{3}{\underset{\wedge}{b}}; b = \overset{4}{\underset{\wedge}{y}} \Rightarrow A(a, b) = A(x, y)$$

$x := x + 1 \Rightarrow a = a + 1 = 4$  estos cambios se  
 $y := y - 1 \Rightarrow b = b - 1 = 3$  hacen a medida que corre el código

la función que de modificar la variable de la función que llamar.

Si dentro de A, x e y modifican sus valores, como están referenciando a a y b, estos también modificarán sus valores.

- **Pasaje por valor+resultado:** intenta usar los beneficios del pasaje por referencia sin aliasing (doble referenciamiento a una variable).

Al final del procedimiento/función, se copian las variables locales a los argumentos. **por valor + asignación al final**  
ejemplo anterior:

$$a = 3; b = 4 \Rightarrow A(a, b) = A(3, 4)$$

$$x := x + 1 \Rightarrow x = 4;$$

$$y := y - 1 \Rightarrow y = 3; \text{ FIN}$$

$$a = 4; b = 3$$

- **Pasaje por nombre:** en el cuerpo de la función se sustituye textualmente el argumento para cada instancia de su parámetro

ejemplo anterior:

fun A (x, y) :  $\Rightarrow A(a, b) :$

$$x := x + 1; y := y - 1 \quad a := a + 1; b := b - 1$$

$$a = 3; b = 4$$

$A(a, b)$

parecido (casi igual)  
que por referencia.

● **Pasaje por necesidad:** es pasaje por nombre zero con memorización.

Cuando se pasa el argumento, no solo se sustituye el parámetro sino que la primera vez que se calcula, se guarda y se memoriza para el resto de las ejecuciones en ese bloque.

Ejemplo anterior:

fun A (x, y):  
  x := x + 1; y := y - 1;  
  x := y + z;  
  a = 3; b = a + 1  
  A(a, b)

A(a, b)  
a := a + 1; b := b - 1  
(y) (a + 1 - 1 = 3)  
a = b + z  
(3 + z)

no vuelve a calcular a + 1

Se guarda



Tiempo en el que un identificador de variable está ligado a una dirección de memoria.

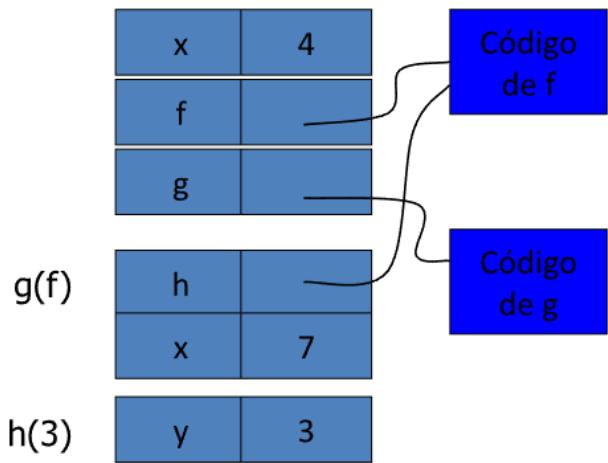
**Alcance estático:** el valor de las variables libres se obtiene del bloque contenedor, es decir, del contexto de definición de la función.

**Alcance dinámico:** el valor de las variables libres se obtiene del activation record anterior, es decir, del contexto de ejecución de la función.

## ejemplo (filminas)

main

```
...  
val x = 4;  
fun f(y) = x*y;  
fun g(h) =  
let  
    val x=7  
in  
    h(3) + x;  
g(f);
```



alcance :  $x=4$ , que se obtiene el valor **estático** del contexto de definición de la función (bloque conenedor)

alcance :  $x=7$ , que se obtiene el valor **dinámico** del contexto de ejecución de la función (activation record más cercano)

Es decir, si estoy en **alcance estático** me pregunta: ¿Dónde defini la función que estoy ejecutando? ¿En qué bloque? Y de ahí sacaré el valor de la variable libre.

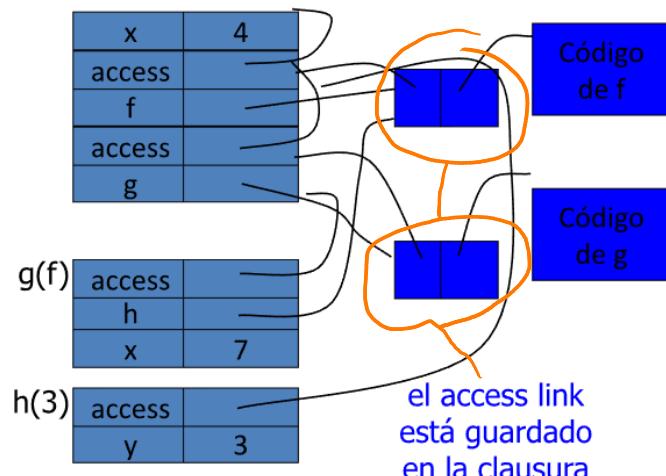
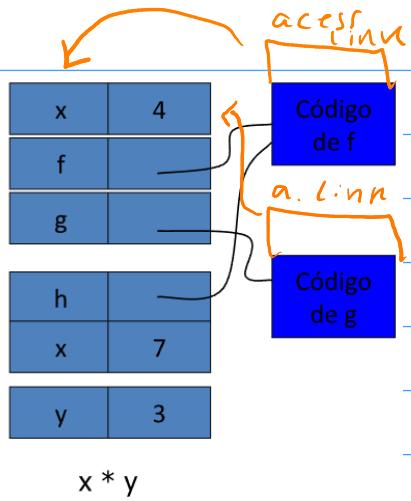
Si estoy en **alcance dinámico** me pregunta: ¿Dónde ejecuté la función? ¿En qué bloque? Y de ahí sacaré el valor de la variable libre.

## Los activation records tendrán un campo que será el access link

records tendrán

un campo que será el access link

→ apunta al contexto = en el que definió la función



Clausura: todo lo que hace a la semántica de la función  
 clausura = (entorno, código)

## Excepciones

Permiten terminar una parte de la ejecución ante un comportamiento inesperado:

- Saltar fuera de una construcción
- pasar datos como parte del salto
- retornar al lugar más reciente que lanza la excepción
- se van desagilando los activation records hasta llegar a donde queda tratarse la excepción
- ↳ los manejadores de excepción tienen alcance dinámico

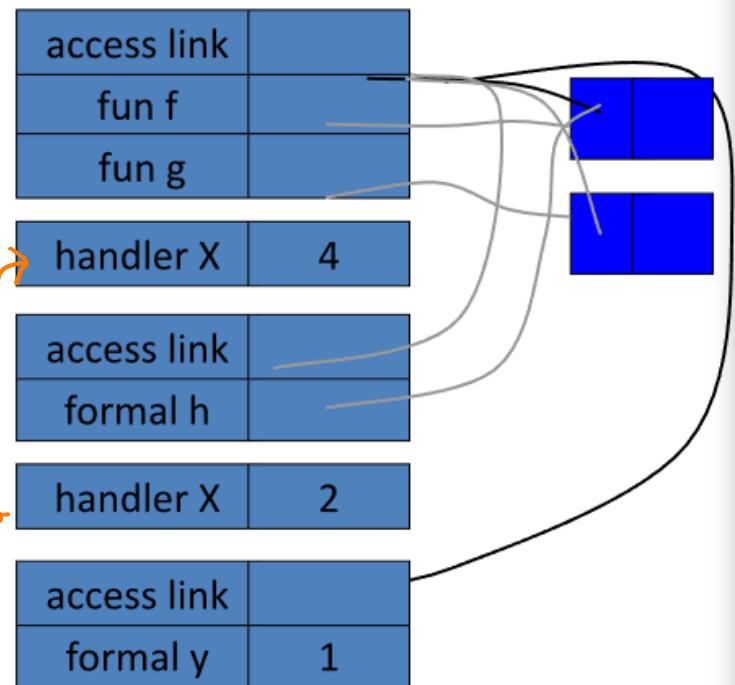
Comprender por dos construcciones:

- un **manejador** que captura y trata
- otra levanta o **tira** la excepción para que sea manejada

Ejemplo (Filminas)

```
exception X;  
fun f(y) = raise X  
fun g(h) = h(1) handle X => 2  
g(f) handle X => 4
```

Cuando voy a ejecutar una función, primero agilo el activation record para el handler así cuando salte la excepción y desagile llegare a ese handler



\* Si hay un **Finally** en la construcción que maneja excepciones **SIEMPRE** se ejecuta, por lo que debe agilarse encima del handler

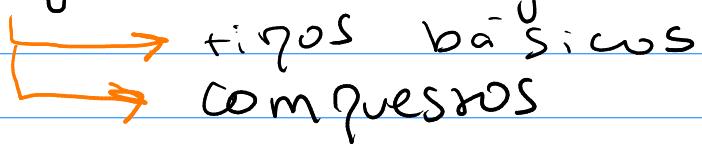
# Tipado

Colección de valores que comparten alguna propiedad estructural.

- Organizan y documentan programas.
- Ayudan a identificar y prevenir errores

- Optimizan el espacio en memoria.

Se organizan en jerárquicas



Los lenguajes se pueden clasificar según su tipado:

Tipado estático vs. dinámico

se fijan los tipos en tiempo de compilación

se fijan los tipos en tiempo de ejecución

Tipado fuerte vs débil

• Se detectan TODOS los errores de tipo

- las conversiones deben ser explícitas
- brindan más seguridad

- el lenguaje puede castear por si solo si es necesario
- brindan más expresividad

Hay tipos primitivos  
↳ booleanos  
↳ enteros  
↳ caracteres  
↳ reales

y construidos

↳ enumerados  
↳ tuplas  
↳ listas  
↳ arreglos

Sobrecarga: sucede cuando el significado de un operador o función depende de los tipos de sus argumentos.  
Un solo símbolo se refiere a más de un algoritmo.

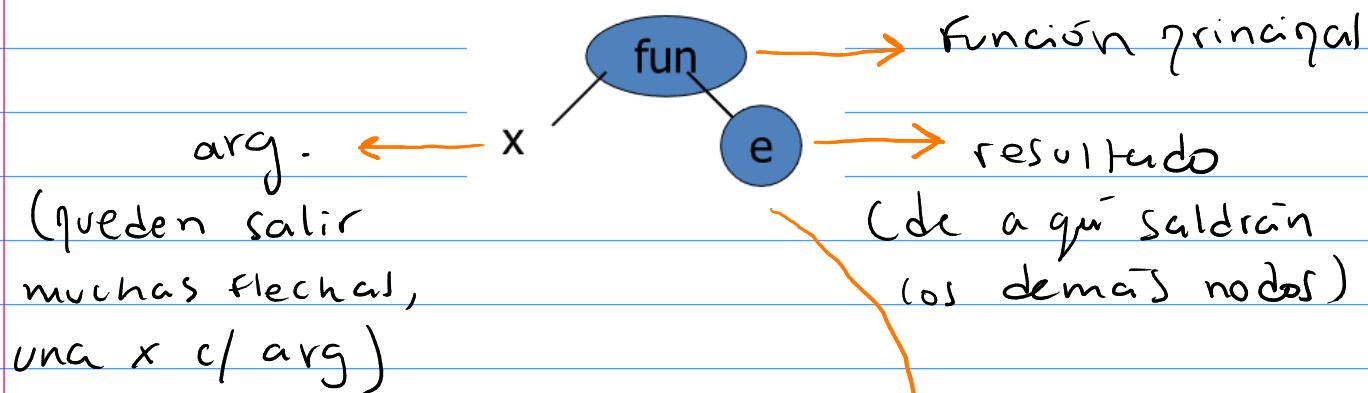
Polimorfismo: un operador o función es polimórfico si se puede aplicar a cualquier tipo relacionado.  
Un solo algoritmo tiene diferentes tipos.

Inferencia de tipos: mirar el código sin información de tipo y figurarse qué tipos se podrían haber declarado.

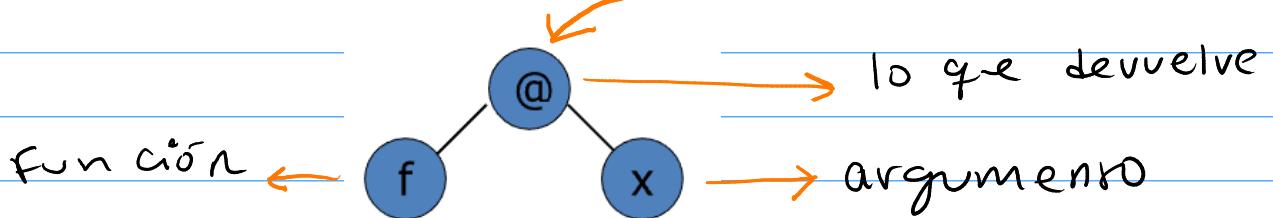
Pasos:

- 1 - Se forma un grafo de las expresiones a tipar.
- 2 - Se asignan tipos a los nodos
- 3 - se generan restricciones
- 4 - se resuelve por sustitución

1- estos serán los nodos  
definición de la función

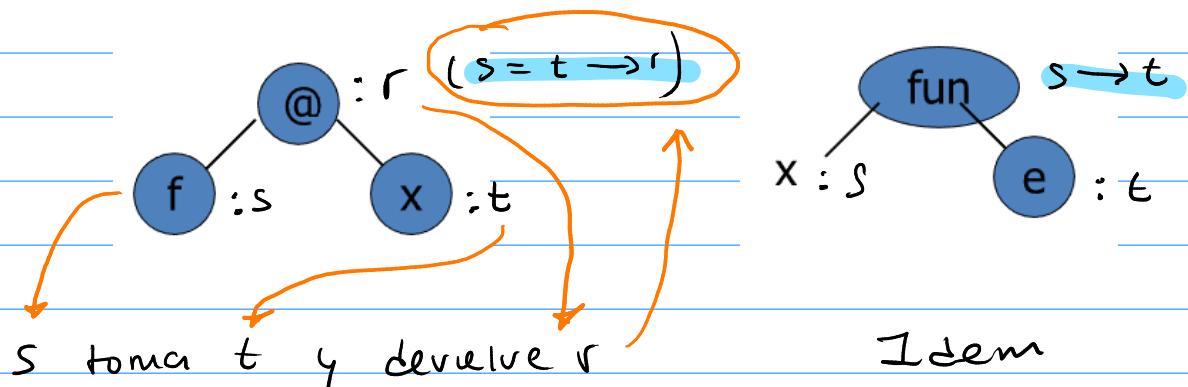


aplicación de la función



Todas las funciones son concatenación de funciones unarias! Por eso nos sirven estos nodos.

2- Agrego los que sé conozco y los que no (estos como incognitivos)



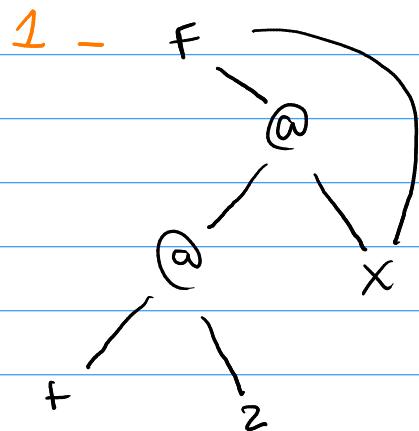
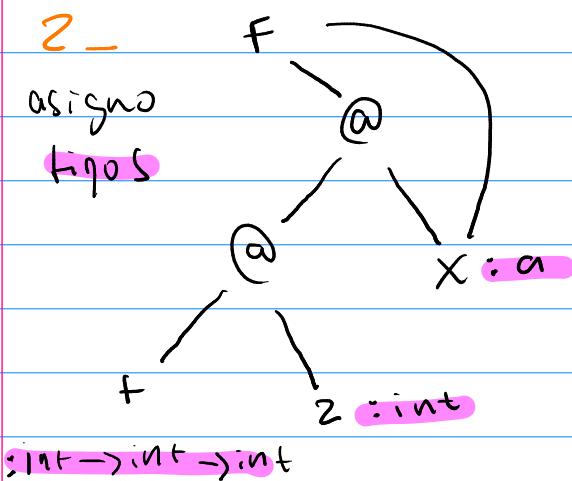
3 - generamos esas restricciones

4 - Resolvemos por sustitución

ejemplo (filminas)

$$f(x) = 2 + x$$

2 -  
asigno  
tipos

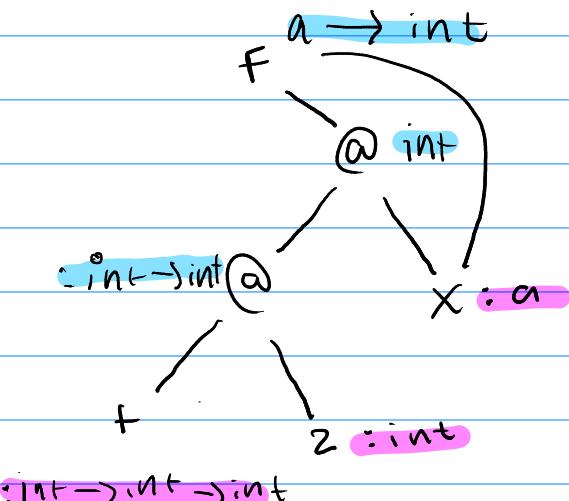


3 - añado restricciones

4 - resuelvo por  
sustitución

$a = \text{int}$

$$\Rightarrow f : \text{int} \rightarrow \text{int}$$



# Práctica

## P.declarativa

9.1. Cuál de estas dos funciones, rq o rt, es transparente referencialmente?

```
1 globalValue = 0;  
2  
3 integer function rq(integer x)  
4 begin  
5     globalValue = globalValue + 1;  
6     return x + globalValue;  
7 end  
8  
9 integer function rt(integer x)  
10 begin  
11     return x + 1;  
12 end
```

rt es transparente referencialmente  
ques  $rt(x) = rt(y)$  ya que el valor de  
la función no depende de ninguna  
otra variable o función por fuera de  
ella.

9.2. El siguiente pedazo de código es declarativo? Por qué?

```
1 { int x = 1;           asignación => no es declarativo  
2   x = x+1;          destrucción  
3   { int y = x+1;      ↑  
4     { int x = y+z;      ↓  
5   } } }
```

9.3. Si una componente de software usa una variable global en una guarda (como parte de una condición con "if"), entonces no es independiente de contexto. Si, en cambio, una componente de software modifica una variable global pero ésta no afecta a su ejecución, conserva la transparencia referencial pero puede tener efectos secundarios.

Escriba dos programas en pseudocódigo, uno en el que se de el primer fenómeno (una variable global que hace que un programa no sea independiente de contexto, es decir, que los resultados de su ejecución no dependan solamente de sus parámetros de entrada), y otro en el que se de el segundo fenómeno (un programa que tiene efectos secundarios a través de una variable global).

variable global = 0

```

func depDeCtx (x) :
    varGlobal = varGlobal + 1
    if (varGlobal < 5) then
        return x + varGlobal
    else
        return varGlobal - x
}

```

No hay transparencia referencial

varGlobal = 0

```

func transRef (x)
    if x > 5 then
        varGlobal = varGlobal + 1
    else
        varGlobal = varGlobal - 1
    return x
}

```

(hay transf. ref.)

9.5. En los siguientes programas imperativos, identifique porciones de código que no sean declarativas.

```

1   int A;
2   int B;
3
4   int Add()
5   {
6       return A + B;
7   }
8
9   int main()
10  {
11      int answer;
12      A = 5;
13      B = 7;
14      answer = Add();
15      printf("%d\n", answer);
16      return 0;
17  }
}

```

que usa A y B "globales",  
no es independiente  
del contexto.

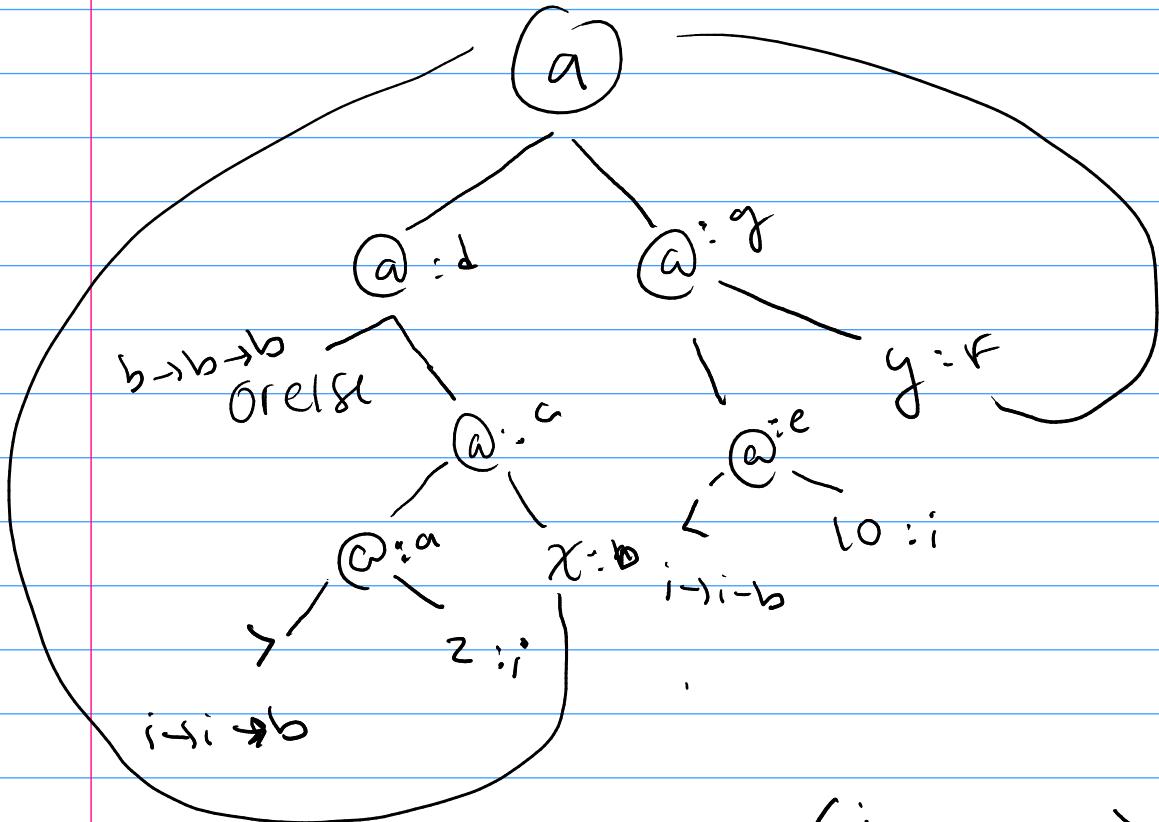
```
1 int glob = 0;
2
3 int square(int x)    } tiene efectos secundarios
4 {                      } en el resto del programa
5   glob = 1;
6   return x*x;
7 }
8
9 int main()
10 {
11   int res;
12   glob = 0;
13
14   res = square(5);
15   res += glob;      → aquí glob = 1 entonces
16
17   return res;
18 }
```

↓

aquí glob = 1 entonces  
cambia el valor de res.

## Inf. de tipos

$$\textcircled{1} \quad \text{fun } a(x, y) = \left( \left( \begin{matrix} x > z \\ \uparrow \end{matrix} \right) \text{ or else } \left( \begin{matrix} y < l_0 \\ \uparrow \end{matrix} \right) \right)$$



$<:$        $>:$

$$\textcircled{1} \text{ int} \rightarrow \text{int} \rightarrow \text{bool} = \text{int} \rightarrow a \circ \text{int} \rightarrow e$$

$$\textcircled{2} \text{ bool} \rightarrow \text{bool} \rightarrow \text{bool} = c \rightarrow d$$

$$\textcircled{3} \quad a = b \rightarrow c \quad \textcircled{4} \quad e = f \rightarrow g$$

$$\textcircled{5} \quad d = g \rightarrow \text{fun}$$

por  $\textcircled{1}$  tenemos que  $a = \text{int} \rightarrow \text{bool}$ ;  $e = \text{int} \rightarrow \text{bool}$

por  $\textcircled{3}$  y  $\textcircled{4}$  tenemos que  $b = \text{int}$ ;  $f = \text{int}$

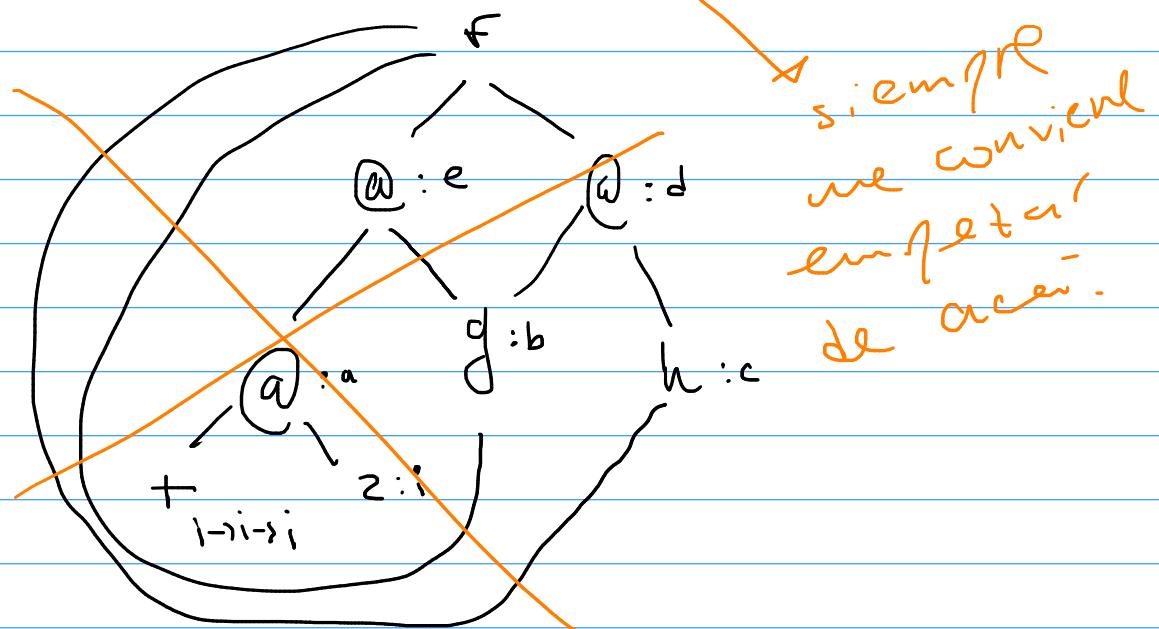
$c = \text{bool}$  y  $g = \text{bool}$

por  $\textcircled{2}$  tengo  $\text{bool} \rightarrow \text{bool} \rightarrow \text{bool} = \text{bool} \rightarrow d$

y entonces  $d = \text{bool} \rightarrow \text{bool}$  y  $\textcircled{5}$  queda

$\text{bool} \rightarrow \text{bool} = \text{bool} \rightarrow \text{fun} \Rightarrow \text{fun} : \text{int} \rightarrow \text{int} \rightarrow \text{bool}$

$$\textcircled{2} \quad f(g, h) = g(h) + z$$

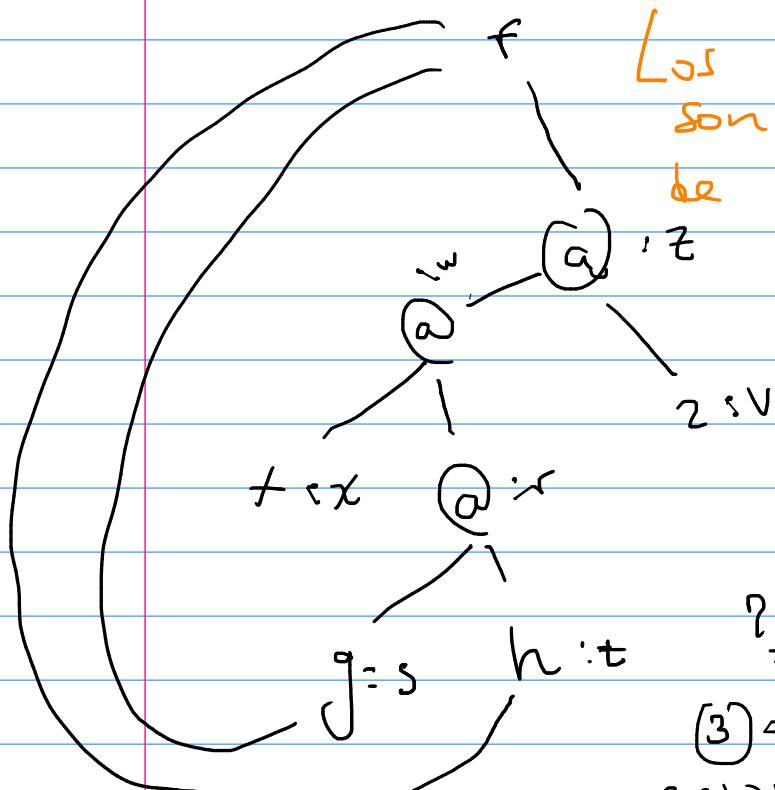


\( \textcircled{1} \) int \(\rightarrow\) int \(\rightarrow\) int = int \(\rightarrow\) a

\( \textcircled{2} \) a = b \(\rightarrow\) e    \( \textcircled{3} \) b = c \(\rightarrow\) d

\( \textcircled{4} \) e = d \(\rightarrow\) f

por \(\textcircled{1}\) a = int \(\rightarrow\) int, entonces \(\textcircled{2}\) \(\Rightarrow\) b = int y e = int



Los tipos que debemos calcular  
son los de los argumentos  
de f. (g y h)

\( \textcircled{1} \) x = int \(\rightarrow\) int \(\rightarrow\) int

\( \textcircled{2} \) v = int (\(\textcircled{3}\)) s = t \(\rightarrow\) r

\( \textcircled{4} \) x = r \(\rightarrow\) w (\(\textcircled{5}\)) w = v \(\rightarrow\) z

\( \textcircled{6} \) f = s \(\rightarrow\) t \(\rightarrow\) z

por \(\textcircled{1}\) y \(\textcircled{4}\) \(\Rightarrow\) r \(\rightarrow\) w = int \(\rightarrow\) int \(\rightarrow\) int

por \(\textcircled{5}\) w = int \(\rightarrow\) int ; por \(\textcircled{2}\)  
z = int ; por \(\textcircled{1}\) r = int

\(\textcircled{3}\) \(\Rightarrow\) s = t \(\rightarrow\) int

entonces \(\textcircled{6}\) \(\Rightarrow\) f = (t \(\rightarrow\) int) \(\rightarrow\) t  
↳ int

t no lo queremos saber, e) variable de finito.

# REUPERATORIO 18/06/24

1. En el siguiente programa identifique en qué línea se daría un comportamiento diferente en un lenguaje con alcance estático y en un lenguaje con alcance dinámico [5 pt.] y describa por qué [10 pt.], usando los conceptos de *contexto de definición* y *contexto de ejecución*. Indique qué escribiría el programa en cada uno de esos casos.

```

1 Program A() {
2     x, y, z: integer;
3
4     procedure B() {
5         y: integer;
6         y=0;
7         x=z+1; → qul.
8         z=y+2; → qul
9     }
10
11    procedure C() {
12        z: integer;
13
14        procedure D() {
15            x: integer;
16            x = z + 1;
17            y = x + 1;
18            call B();
19        }
20        z = 5;
21        call D();
22    }
23
24    x = 10; y = 11; z = 12;
25    call C();
26    print x, y, z;
27 }
```

|     |      | Est. | Din..                          |
|-----|------|------|--------------------------------|
| A   | x 10 | 13   | 10                             |
| B   | z 12 | 2    | 12                             |
| C   | y 7  | 7    | 7                              |
| D   |      |      |                                |
| C() |      |      | 2                              |
| B() |      |      | 6                              |
| D() |      |      |                                |
|     |      |      | $p(13, 2, 2) \neq (10, 7, 12)$ |

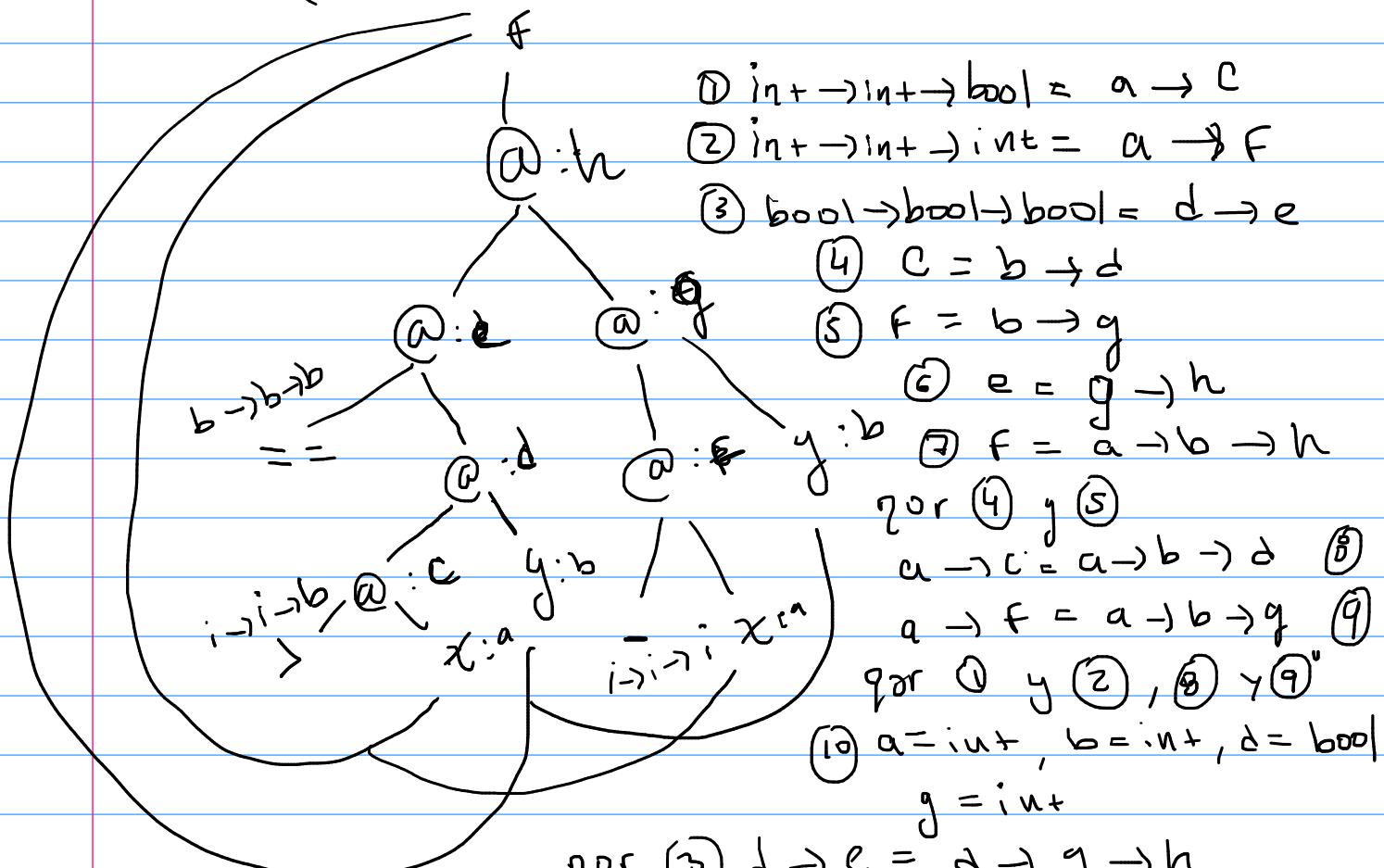
Estativo: (12, 13, 2) ; Dinámico: (10, 7, 12)

Las líneas en las que se presentan comportamientos diferentes dependiendo del alcance son la 7 y la 8, ya que si esramos en alcance estativo modificaremos y usaremos los valores de las variables libres en su contexto de definición, es decir, donde se definió el procedimiento que hace uso de ellas, en este caso, se modifican y usan los valores de x y z definidas en el proc A, que es donde se define el proc B que hace uso y modificación de ellas.

En cambio si estamos en alcance dinámico se usan y modifican la  $x$  y la  $z$  en el contexto de ejecución del proceso que hace uso de ellas, es decir, donde se ejecutó el proc. B. Se usarán y modificarán la  $x$  y la  $z$  del proc. C y D, que ahí se ejecuta B.

2. [10 pt.] La siguiente expresión está mal tipada:  $f(x,y) = x > y == x - y$ . Diagrama el grafo de tipado y el sistema de ecuaciones correspondiente [5 pt.], describa dónde se encuentra el problema y explique cómo lo trataría un lenguaje de tipado fuerte y cómo podría tratarlo un lenguaje de tipado no fuerte.

$$f(x,y) \neq ((x > y) = (x - y))$$



es erróneo!!  $\Rightarrow x > y$  es bool entonces no  
 $x - y$  es int y deben ser  
 los args de  $=$

Un lenguaje de tipado fuerte casearía alguno de los dos valores resultado de  $x > y$  o  $x - y$  para que no salte un error de tipo.

Un lenguaje de tipos fuerte devolvería un error y no compilaría (si es tipado estático) o detendría la ejecución del programa (si es tipado dinámico)

3. [10 pt.] Qué imprime el siguiente programa con pasaje de parámetros por valor y con pasaje de parámetros por referencia? Diagrama los estados por los que pasa la pila de ejecución en cada uno de los casos, de forma que se vea claramente cómo se modifican las variables en cada caso.

```

1 int x=0;
2 p (int ,int );
3 main () {
4     int x = 1;
5     p (x,x);
6 }
7 p (int y, int z){
8     x = x+1;
9     y = y+1;
10    z = z+1;
11    print (x+y+z);
12 }
```

### P. por valor

#### Momento 1

|         |   |
|---------|---|
| X       | 0 |
| ? (y,z) | - |

#### Momento 2

|         |   |
|---------|---|
| X       | 0 |
| ? (y,z) | - |

#### Momento 3

|         |   |
|---------|---|
| X       | 0 |
| ? (y,z) | - |

|         |   |
|---------|---|
| X       | 1 |
| ? (i,i) | - |

|   |   |
|---|---|
| y | 1 |
| z | 1 |
|   | 1 |

|         |   |
|---------|---|
| X       | 2 |
| ? (i,i) | - |

|   |   |
|---|---|
| y | 2 |
| z | 2 |
|   | 2 |

print (2+2+2)

# P. por referencia

Momento 1

|   |        |
|---|--------|
| X | 0      |
| ? | (y, z) |

Momento 2

|   |        |
|---|--------|
| X | 0      |
| ? | (y, z) |

Momento 3

|   |        |
|---|--------|
| X | 1      |
| ? | (y, z) |

Momento 4

|   |        |
|---|--------|
| X | 1      |
| ? | (y, z) |

main()

|   |        |
|---|--------|
| X | 2      |
| ? | (x, x) |
| y |        |
| z |        |

Momento 5

|   |        |
|---|--------|
| X | 1      |
| ? | (y, z) |

|   |        |
|---|--------|
| X | 3      |
| ? | (x, x) |
| y |        |
| z |        |

$$\text{print}(3+3+1) \Rightarrow 7$$

4. [10 pt.] Identifique dos propiedades no declarativas en la siguiente componente, y describa por qué son no declarativas.

```
1 def calculate_sum(numbers):
2     for num in numbers:
3         sum += num → no declarativa
```

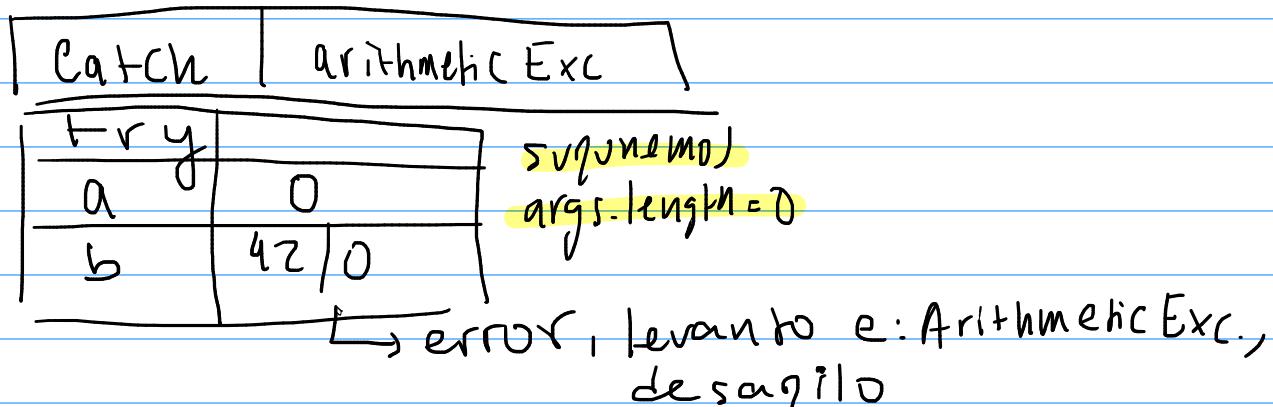
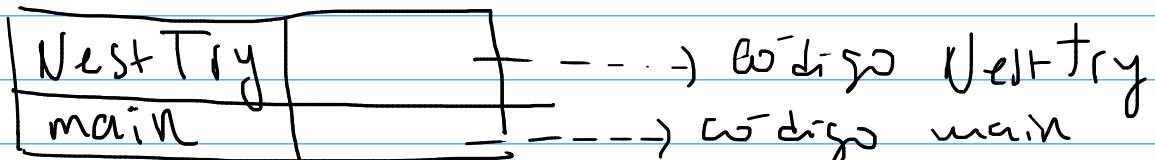
La línea 3 no es declarativa ya que se realiza una asignación destructiva, se sobreescribe la variable sum en cada iteración del for.

5. [15 pt.] Describa cómo es la ejecución del siguiente programa con el detalle de qué activation records se van apilando y desapilando. Puede complementar su explicación con diagramas de la pila de ejecución si le resulta más claro.

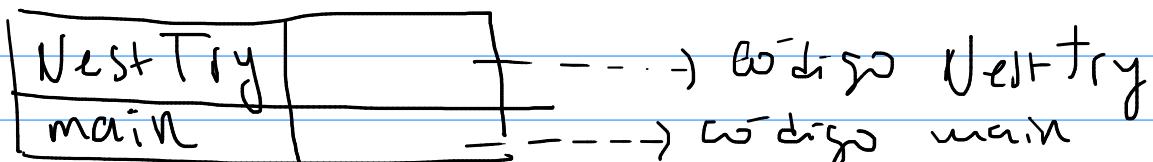
```

1 class NestTry {
2     public static void main(String args[]) {
3         try {
4             int a = args.length;
5             int b = 42 / a;
6             System.out.println("a = " + a);
7             try {
8                 if (a==1) a = a/(a-a);
9                 if (a==2) {
10                     int c[] = { 1 };
11                     c[42] = 99;
12                 }
13             } catch (ArrayIndexOutOfBoundsException e) {
14                 System.out.println("Array index out-of-bounds: " + e);
15             }
16             } catch (ArithmeticException e) {
17                 System.out.println("Divide by 0: " + e);
18             }
19     }
20 }
```

línea  
de la  
1 a la  
5



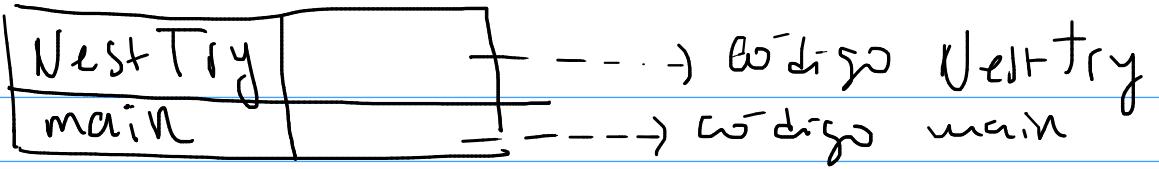
líneas  
16 y 17



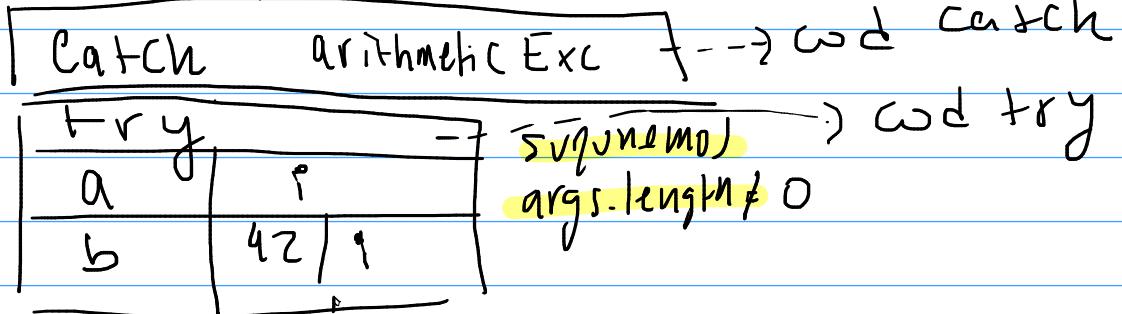
|       |                |
|-------|----------------|
| Catch | arithmetic Exc |
|-------|----------------|

print("divide by 0: " + e)

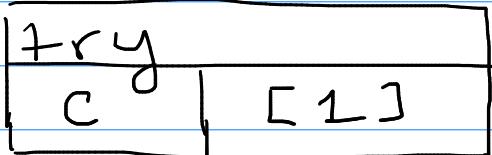
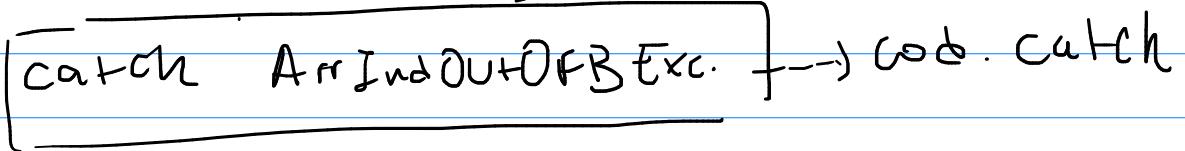
lineas  
de la  
lata



12

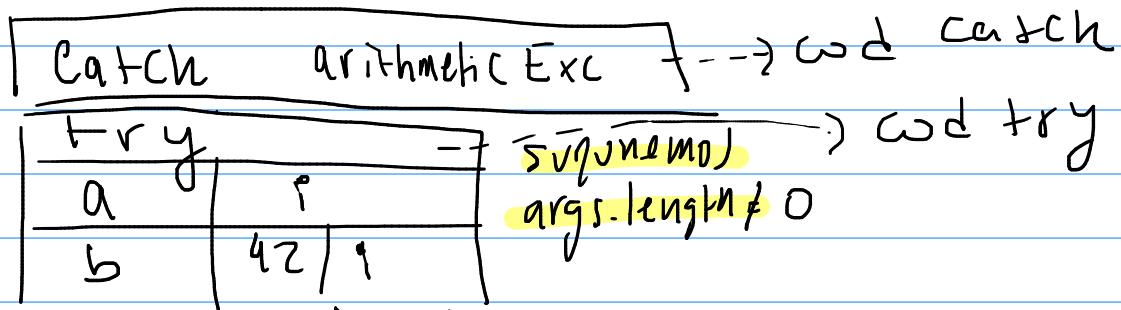
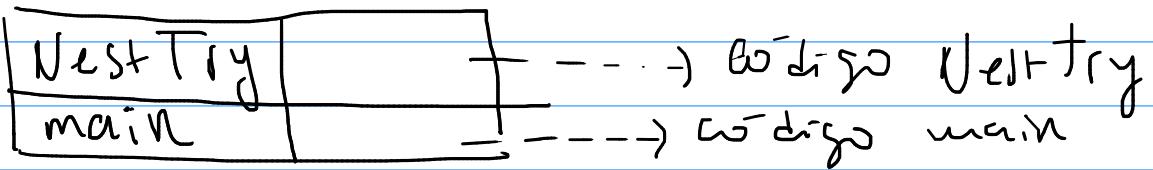


```
print('a = ' + a)
```

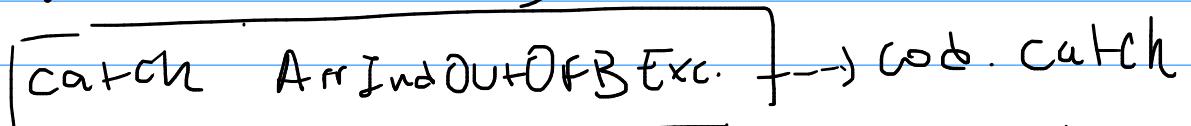


accede a  $C[42]$  → error, evento e:

ArrayIndexOutOfBoundsException  
desapilado



Print(̂a = ' + a)



print('Array index out-of-bounds!')

# RECUERATORIO 11/06/19

1. Dado el siguiente programa:

- [10 pt.] Diagrama los estados por los que va pasando la pila de ejecución al ejecutarse el programa, asumiendo que el lenguaje tiene alcance estático.
- [10 pt.] ¿Qué se imprimiría si el lenguaje tiene alcance estático? ¿Y si tiene alcance dinámico?
- [10 pt.] ¿Qué se imprimiría si el lenguaje tiene pasaje de parámetros por valor? ¿Y si tiene pasaje de parámetros por referencia?

```

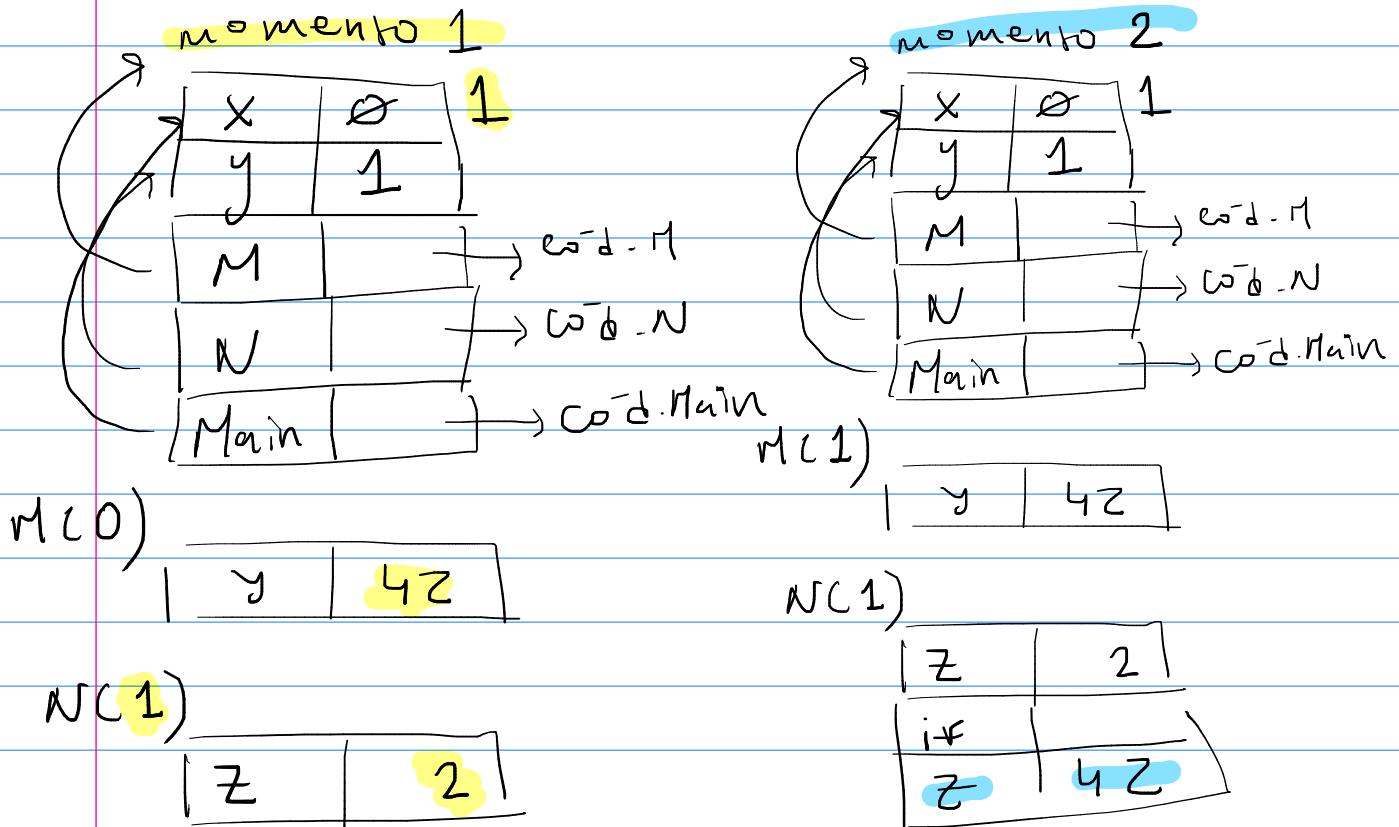
var x : int = 0;
var y : int = 1;

procedure M(x : int)
begin
    var y : int = 42;
    x := 1;
    call N(x);
end

procedure N(x : int)
begin
    var z : int = 2;
    if true begin
        var z : int = 42;
    end
    print x;
    print y;
    print z;
end

procedure Main()
begin
    call M(x);
end

```



momento 3

|      |   |              |
|------|---|--------------|
| x    | 0 | 1            |
| y    | 1 |              |
| M    |   | → es-d.-M    |
| N    |   | → w-b.-N     |
| Main |   | → co-d.Main. |

b) En alcance  
estáticos imprime  
 $x = 1, y = 1,$   
 $z = z$   
y en dinámico  
 $x = 1, y = 42,$   
 $z = z$

M(1)

|   |    |
|---|----|
| y | 42 |
|---|----|

N(1)

|   |   |
|---|---|
| z | 2 |
|---|---|

print 1  
print 1  
print 2

c) Pasaje por valores:

M(0)

|   |    |
|---|----|
| y | 42 |
|---|----|

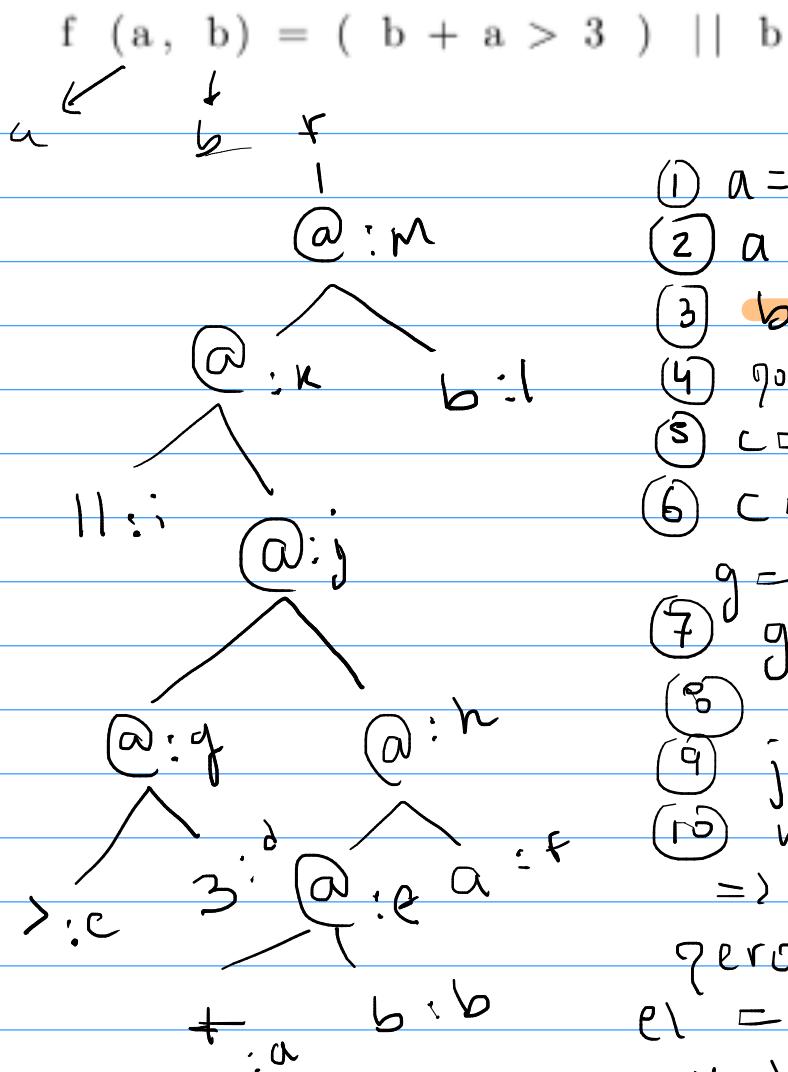
N(1)

|       |   |
|-------|---|
| z     | 2 |
| print | 1 |
| print | 1 |
| print | 2 |

Por ref:

print 1  
print 1  
print 2

2. [10 pt.] La siguiente expresión está mal tipada:

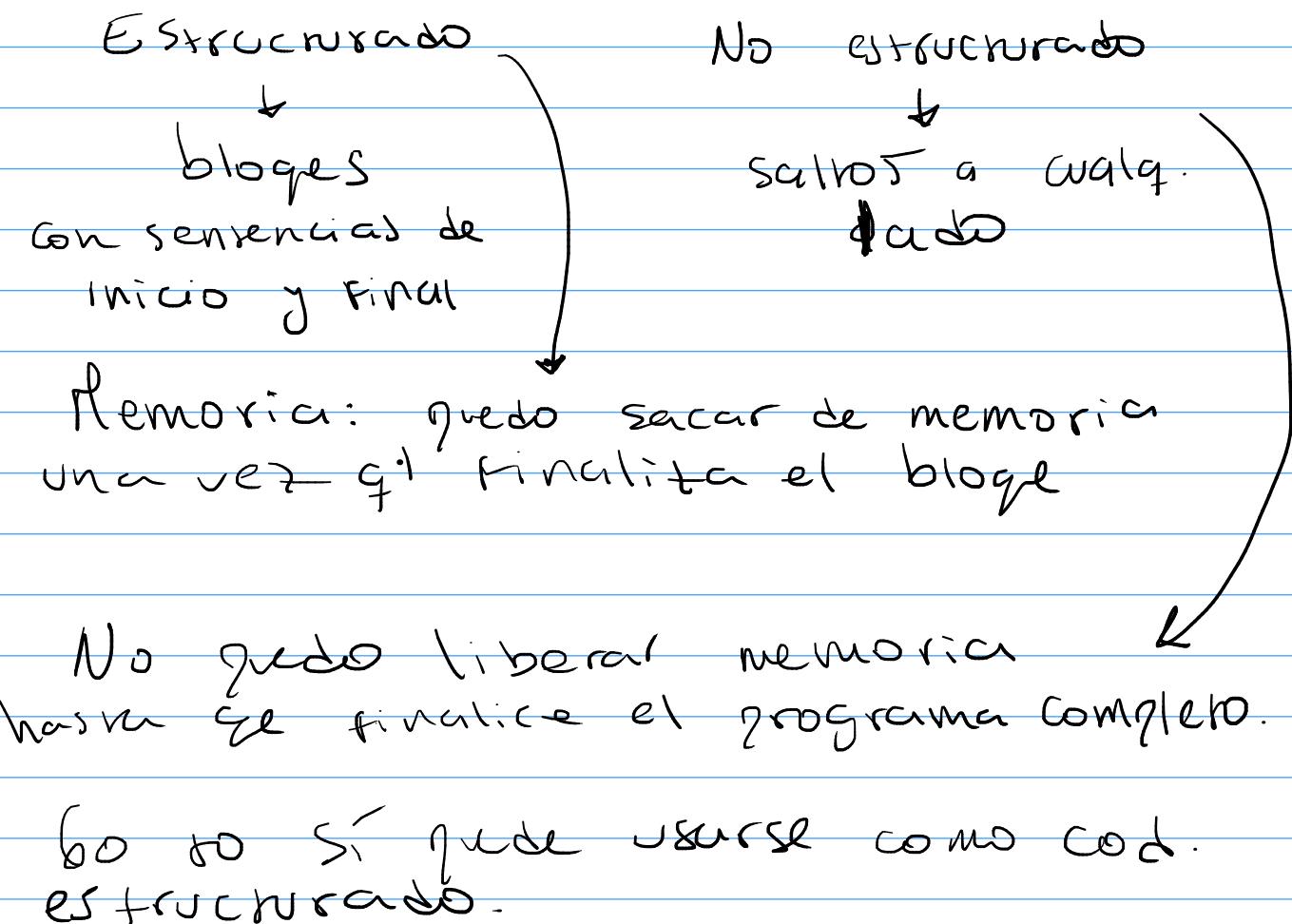


- (1)  $a = \text{int} \rightarrow \text{int} \rightarrow \text{int}$
- (2)  $a = b \rightarrow e$
- (3)  $b = \text{int} \cdot e = \text{int} \rightarrow \text{int}$
- (4)  $\text{or } (3) \quad f' = \text{int} \wedge h = \text{int}$
- (5)  $c = \text{int} \rightarrow \text{int} \rightarrow \text{bool}$
- (6)  $c = d \rightarrow g \wedge d = \text{int} \wedge g = \text{int} \rightarrow \text{bool}$
- (7)  $g = h \rightarrow j, h = \text{int}, j = \text{bool}$
- (8)  $i = \text{bool} \rightarrow \text{bool} \rightarrow \text{bool}$
- (9)  $j = \text{bool}, k = \text{bool} \rightarrow \text{bool}$
- (10)  $k = l \rightarrow m$   
 $\Rightarrow l = \text{bool}, m = \text{bool}$   
zero  $l = b$   $z\circ r g e \ s o n$   
 $e1 = \text{parámetro}$   
 $y \text{ bool} \neq \text{int}$   
(1) (b)

MAL TIPOADO  $\Leftarrow$  ABSURDO !!

Típado fuerte: tira errores  
débil: se casrea algunas  
de las variables para evitar  
el error.

3. [10 pt.] Explique por qué la instrucción `go to` puede producir código no estructurado. Explique a partir de eso cuál es la diferencia esencial entre código estructurado y no estructurado, y qué impacto tiene esa diferencia en la gestión de la memoria. La instrucción `go to`, ¿puede usarse siguiendo las reglas de código estructurado?



4. [10 pt.] Teniendo en cuenta que la siguiente función en Javascript tiene transparencia referencial ¿qué diferencia hay entre pasar los parámetros por valor y pasarlo por referencia?

```
const sum = a => b => a + b;  
console.log (sum (5) (3));
```

La f. ref.  $\Rightarrow$  No hay diferencia en el manejo por valor y por referencia, ya que asegura determinismo en el programa y que ese no tengan efectos secundarios fuera de sí.

5. [10 pt.] Estas dos funciones tienen la misma semántica, pero una de ellas no es declarativa. ¿Cuál no es declarativa y por qué no lo es?

```
const container = document.getElementById('container');
const btn = document.createElement('button');
btn.className = 'btn red';
btn.onclick = function(event) {
  if (this.classList.contains('red')) {
    this.classList.remove('red');
    this.classList.add('blue');
  } else {
    this.classList.remove('blue');
    this.classList.add('red');
  }
};
container.appendChild(btn);
```

acá hay asignación  
destrucción, ya que  
modificamos classList y  
No es independiente ya  
que el comportamiento depen-  
del orden de los listas.

```
class Button extends React.Component{
  this.state = { color: 'red' }
  handleChange = () => {
    const color = this.state.color === 'red' ? 'blue' : 'red';
    this.setState({ color });
  }
  render() {
    return (<div>
      <button
        className='btn ${this.state.color}'
        onClick={this.handleChange}>
      </button>
    </div >);
  }
}
```

7. [10 pt.] Diagrama los estados por los que pasa la pila de ejecución en el siguiente programa en java, enfocándose únicamente en el manejo de excepciones, sin representar las variables locales, control links, access links, retorno de función ni ninguna otra información que no sea relevante al proceso de manejo de excepciones. Explique qué pasa con los dos catch, y por qué.

```

class Test
{
    public static void main(String [] args)
    {
        try
        {
            int a[] = {1, 2, 3, 4};
            for (int i = 1; i <= 4; i++)
            {
                System.out.println ("a[" + i + "]=" + a[i] + "n");
            }
        }
        catch (Exception e)
        {
            System.out.println ("error = " + e);
        }
        catch (ArrayIndexOutOfBoundsException e)
        {
            System.out.println ("ArrayIndexOutOfBoundsException");
        }
    }
}

```

Test

main

main()

catch | ArrayIndexOutOfBoundsException

catch | Exception

try

print "a[" + i + "]=" + a[4] + "n")

↳ error,

levanto la excepción para  
que la aguere algún handler (catch) q  
empiecto a desaqilar

Test

main

main()

catch | ArrayIndexOutOfBoundsException

catch | Exception

→ llego a esta

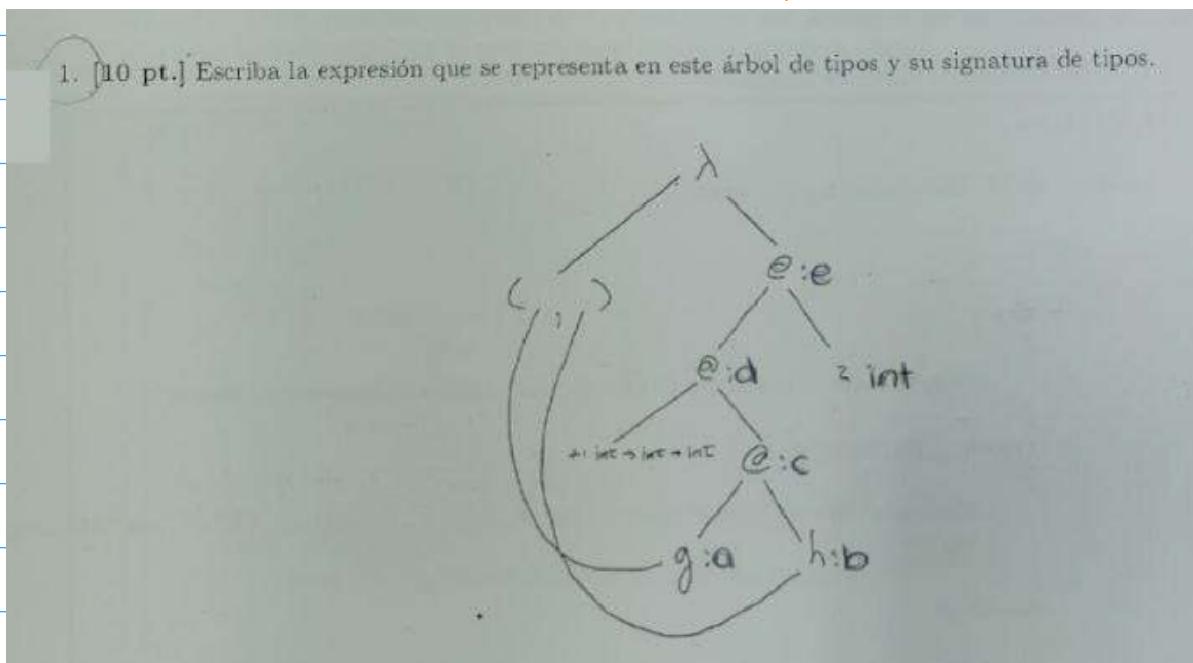
excepción primero,

como es general es ella quien la  
maneja

print ("error = " + e)

→ excepción

# PARCIAL 9/5/23



$$\lambda(g,h) = g(h) + z$$

$h \Rightarrow$  variable de tipo (no se sabe qué tipo tendrá)

- (1)  $a = b \rightarrow c$
- (2)  $\text{int} \rightarrow \text{int} \rightarrow \text{int} = c \rightarrow d$
- (3)  $d = \text{int} \rightarrow e$
- (4)  $d : a \rightarrow b \rightarrow e$

por (3) y (2)  $\text{int} \rightarrow \text{int} \rightarrow \text{int} = c \rightarrow \text{int} \rightarrow e$

entonces  $c = \text{int}$  y  $e = \text{int}$  (5)

por (5)  $a = b \rightarrow \text{int}$  (6) y ahora sí

por (4) y (6)  $d : (b \rightarrow \text{int}) \rightarrow b \rightarrow \text{int}$

2. [20 pt.] En el siguiente programa, ¿encontraremos una diferencia si el alcance del lenguaje es estático o si el alcance es dinámico? ¿Qué se imprimiría en cada caso?

```

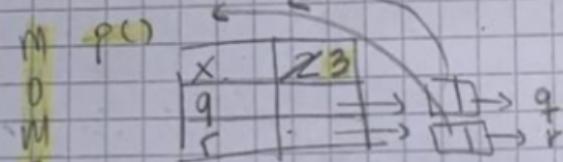
1 procedure p;
2   x: integer;
3   procedure q;
4     begin x := x+1 end;
5   procedure r;
6     x: integer;
7     begin x := 1; q; write(x) end;
8   begin
9     x:= 2;
10    r;
11  end;

```

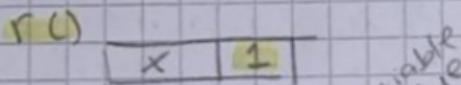
¡Usa esa a!

⑤ Alcance estático:

las cajitas de esa forma tienen esa estructura

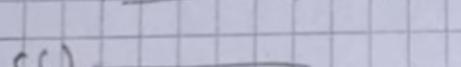
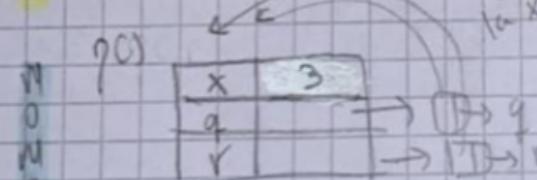


access | código  
l:nc | a la función  
que señalan

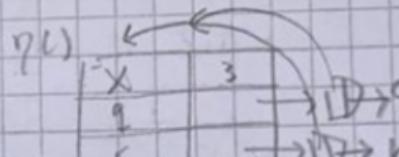


variable  
libre

(modifica la x en  
contrario al definición de  
la x en p)



write(1)



M  
D  
M  
P  
N  
O  
1

## Alcance dinámico:

M q()

|   |   |
|---|---|
| X | z |
| q |   |
| r |   |

(no necesario acceder link)

M r()

|   |   |
|---|---|
| x | z |
|   |   |

M g()

modifico la  $x$  en el contexto en el que se ejecuta  $g$ , es decir, el contexto de  $r()$

M q()

|   |   |
|---|---|
| X | z |
| q |   |
| r |   |

$\rightarrow \square \rightarrow q$

M r()

|   |   |
|---|---|
| X | z |
|   |   |
|   |   |

write(z)

M p()

|   |   |
|---|---|
| M | 0 |
| O |   |
| M | 3 |
| P |   |
| N |   |
| R |   |
| D |   |

|   |   |
|---|---|
| X | z |
| q |   |
| r |   |

$\rightarrow \square \rightarrow q$

En alcance estático se imprimirá 1, en alcance dinámico se imprimirá 2.

⑥ Pasaie por valor:

dado nos alcance dinámico

3. [20 pt.] Escriban un programa en pseudocódigo en el que

- se define una función que toma una lista y un entero como parámetros, modifica el primer elemento de la lista y devuelve la lista modificada.
- se declaran e inicializan una lista  $l = [1, 2, 3]$  y un entero  $e = 5$ , y declaran una lista  $ll$  que no inicializan
- llaman a la función que modifica la lista, con los argumentos  $l$  y  $e$ , y asignan el resultado de la función a la lista  $ll$
- imprimen  $l$  y  $ll$

Este programa, ¿imprimirá resultados diferentes para pasaje de parámetros por valor, por referencia, por valor-resultado, por nombre y por necesidad? Diga qué imprimirá para los diferentes tipos de pasaje de parámetros.

a)  $\text{modFirst} (T[]; \text{int}, x: \text{int}) \{$

$T[0] = x$

$\text{return } T$

b)  $l[]: \text{int}$

$e: \text{int}$

$l[] = [1, 2, 3]$

$e = 5$

$ll[]: \text{int} -$

c)  $\} ll[] = \text{modFirst} (l, e)$

d)  $\} \text{print}(l, ll)$

Pasaje por valor:

Llamo a  $\text{modFirst}$  con  $[1, 2, 3]$  y  $5$

$\text{modFirst}(l)$

|        |     |
|--------|-----|
| $T[0]$ | $x$ |
| $T[1]$ | $z$ |
| $T[2]$ | $3$ |
| $x$    | $5$ |

Cuando termina la ejecución desalojo  $\text{modFirst}$  e imprimo

$l = [1, 2, 3]$  porque pasa por valor  
 $ll = [5, 2, 3]$  no me dificulta las variables  
que pase como argumento

## pasaje por referencia

|       |           |
|-------|-----------|
| I [0] | X 5       |
| I [1] | Z 6       |
| I [3] | 3 4       |
| e     | 5         |
| II    | [5, 2, 3] |

Como las variables  
que se modifican  
en modFirst  
hacen referencia  
a las variables  
que pasó como  
args. esas  
se modifican.

modFirst()

|       |  |
|-------|--|
| T [0] |  |
| T [1] |  |
| T [2] |  |
| X     |  |

se imprime  
 $I = [5, 2, 3]$   
 $II = [5, 2, 3]$

## por valor-resultado

Es lo mismo que por valor  
pero al finalizar la ejecución  
de modFirst se copian los  
valores de las variables locales  
en los argumentos.

Entonces

```
print([5, 2, 3], [5, 2, 3])
```

por nombre = idem valor - resultado  
y necesidad y ref.

4. [20 pt.] ¿Qué imprime el siguiente programa?

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         try
6         {
7             try
8             {
9                 throw new Exception("exception_thrown_from_try_block");
10            }
11            catch (Exception ex)
12            {
13                Console.WriteLine("InnerCatchBlockHandling-{0}", ex.Message);
14                throw;
15            }
16            finally
17            {
18                Console.WriteLine("InnerFinallyBlock");
19                throw new Exception("exception_thrown_from_finally_block");
20                Console.WriteLine("This line is never reached");
21            }
22        }
23        catch (Exception ex)
24        {
25            Console.WriteLine("OuterCatchBlockHandling-{0}", ex.Message);
26        }
27        finally
28        {
29            Console.WriteLine("OuterFinallyBlock");
30        }
31    }
32 }
```

Main()

finally

catch(e)

try

finally

catch(e)

try

Se ejecuta este

try, qd hrg

exception\_thrown\_from\_try\_block y se  
desanilla

El catch de arriba la maneja,  
q imprime InnerCatchBlockHandling-{0},  
exception\_thrown\_from\_try y se desanilla.

El finally se ejecuta, imprime Inner-finally-block y lanza una nueva excepción "exception thrown from -finally-block" y se desgila.

Se desgila el try de afuera, que finalizó su ejecución.

El catch de afuera cierra la excepción del finally e imprime Outer-catch-block handling [0], exception-thrown-from-finally-block y se desgila.

Por último se ejecuta el finally de afuera, imprime outer-finally-blocks y se desgila finalizando la ejecución del programa.

5. [10 pt.] Identifiquen en el siguiente programa dos características no declarativas, subráyelas en el código y digan cómo se llaman.

```
1 total = 0
2
3 for i in range(1, 6):
4     square = i * i
5     total += square
6
7 print(total)
```

→ asignación curva vez es no  
destructiva; determinístico  
ya que total depende  
del resultado que tengas  
anteriormente