

1. Explique lo que sucede cuando ejecuta el siguiente programa en MOzArt.

```
declare A B C D in -> Se crean 4 variables A,B,C,D
thread D=C+1 end -> Se crea un hilo que asigna un valor a la variable D
thread C=B+1 end -> Se crea un hilo que asigna un valor a la variable C
thread A=1 end -> Se crea un hilo que asigna un valor a la variable A
thread B=A+1 end -> Se crea un hilo que asigna un valor a la variable B
{Browse D} -> Se muestra en el tablero de Mozart el valor de D (4)
```

a. En qué orden son creados los hilos?

Respuesta: Los hilos se crean en el orden en que están escritos en el código en este caso el hilo de D, el hilo de C, el hilo de A y el hilo de B

b. En qué orden son evaluados los hilos?

Respuesta: Los hilos son evaluados en la manera en que se puede resolver la asignación en este caso la primera asignación que se resuelve es A, luego B, luego C y por ultimo D

2. Comportamiento de flujo de datos en un contexto concurrente. Considere la función

{Filter En F}, que devuelve los elementos de En para los cuales la función booleana F devuelve true. Esta es una posible definición de Filter:

```
fun {Filter En F}
case En
of X|En2 then
if {F X} then X|{Filter En2 F}
else {Filter En2 F} end
else
nil
end
end
```

Al ejecutar la instrucción siguiente:

```
{Show {Filter [5 1 2 4 0] fun {$ X} X>2 end}}
se despliega
[5 4]
```

(Usamos el procedimiento Show, el cual despliega el valor instantáneo de su argumento en la ventana del emulador de Mozart. Es diferente al Browse, pues la salida de Show no se actualiza si el argumento se liga posteriormente.) Entonces Filter se comporta como se espera, en el caso de una ejecución secuencial y que todos los valores de entrada están disponibles. Ahora exploremos el comportamiento de flujo de datos de Filter.

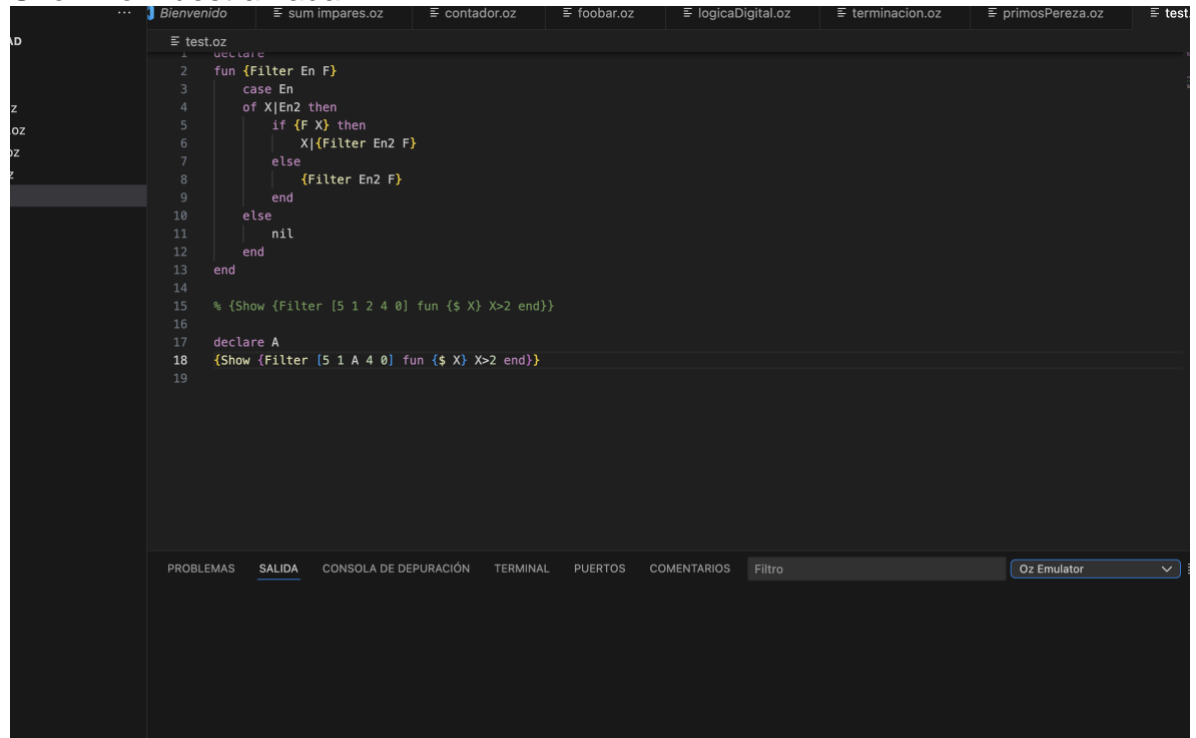
- a. ¿Qué pasa cuando se ejecuta lo siguiente?:

```
declare A
```

```
{Show {Filter [5 1 A 4 0] fun {$ X} X>2 end}}
```

Uno de los elementos de la lista es una variable A que no ha sido ligada aun a ningún valor. Recuerde que las declaraciones case e if suspenderán el hilo en que ellas se ejecutan, hasta que puedan decidir que alternativa tomar.

Respuesta: al llegar a la variable A y no existir una asignación, el programa se suspende para esperar la asignación, si esa asignación nunca se hace, el programa quedara en espera indefinidamente, por lo tanto la función Show no muestra nada.

The screenshot shows the Oz IDE interface. At the top, there are several tabs: 'Bienvenido', 'sum impares.oz', 'contador.oz', 'foobar.oz', 'logicaDigital.oz', 'terminacion.oz', 'primosPereza.oz', and 'test.oz'. The 'test.oz' tab is active. The code editor displays the following code:

```
1 declare
2   fun {Filter En F}
3     case En
4     of X|En2 then
5       if {F X} then
6         X|{Filter En2 F}
7       else
8         {Filter En2 F}
9       end
10    else
11      nil
12    end
13  end
14
15  % {Show {Filter [5 1 2 4 0] fun {$ X} X>2 end}}
16
17  declare A
18  {Show {Filter [5 1 A 4 0] fun {$ X} X>2 end}}
19
```

At the bottom of the IDE, there is a panel with tabs: 'PROBLEMAS', 'SALIDA', 'CONSOLA DE DEPURACIÓN', 'TERMINAL', 'PUERTOS', 'COMENTARIOS', and 'Filtro'. The 'SALIDA' tab is selected, and it shows the text 'Oz Emulator'.

- b. ¿Qué pasa cuando se ejecuta lo siguiente?:

```
declare Sal A
```

```
thread Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end} end
```

```
{Show Sal}
```

Recuerde que invocar Show despliega su argumento tal como esté al momento de la invocación. Se pueden desplegar diversos resultados. ¿Cuáles y por qué? ¿La función Filter es determinística? ¿Por qué sí o por qué no?

Respuesta: El programa muestra `_`, indicando una variable suspendida. Sí se pueden tener varios resultados dependiendo del momento en que se asigne A y el valor que esté tome, pero esos datos no actualizarán el Show si ya se ha ejecutado. Por lo tanto como se tiene el código en estos

momentos no sería posible desplegar varios resultados, dado que no hay otros hilos en ejecución que puedan asignar un valor a A. Filter es determinista, porque a mismos valores de entrada, mismos valores de salida.

```
18 % {Show {Filter [5 1 A 4 0] fun {$ X} X>2 end}}
19
20 declare Sal A
21 thread Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end} end
22 {Show Sal}
23
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

_<optimized>|

c. ¿Qué pasa cuando se ejecuta lo siguiente?:

```
declare Sal A
thread Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end} end
{Delay 1000}
Show Sal}
```

Recuerde que la invocación {Delay N} suspende su hilo por N ms al menos. Durante este tiempo, otros hilos listos pueden ser ejecutados.

Respuesta: La ejecución muestra 5|_ lo que indica una lista inconclusa con variables suspendidas

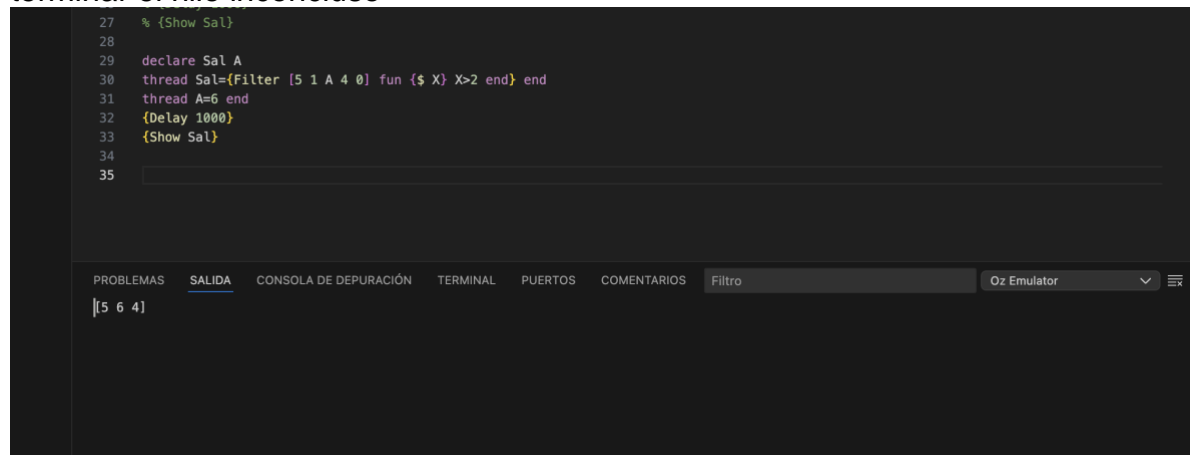
```
21 % thread Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end} end
22 % {Show Sal}
23
24 declare Sal A
25 thread Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end} end
26 {Delay 1000}
27 {Show Sal}
28
29
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS COMENTARIOS Filtro Oz Emulator

5|_<optimized>|

- d. ¿Qué pasa cuando se ejecuta lo siguiente?:
- ```
declare Sal A
thread Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end} end
thread A=6 end
{Delay 1000}
{Show Sal}
```
- ¿Que se despliega y por qué?

**Respuesta:** Se muestra la lista de resultados completa, dado que hay otro hilo que realiza la asignación a la variable A por lo tanto es posible con ello terminar el hilo inconcluso



The screenshot shows a Prolog IDE with a code editor and a terminal window. The code editor contains the following Prolog code:

```
27 % {Show Sal}
28
29 declare Sal A
30 thread Sal={Filter [5 1 A 4 0] fun {$ X} X>2 end} end
31 thread A=6 end
32 {Delay 1000}
33 {Show Sal}
34
35
```

The terminal window shows the output of the code:

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS COMENTARIOS Filtro Oz Emulator
[5 6 4]
```

3. Se crea un productor, un procesador(filtro) y un consumidor.
4. Implemente un programa que registre las ocurrencias de caracteres en un flujo de entrada. El programa debe generar un flujo de salida donde cada posición del flujo corresponde al total de ocurrencias hasta ese momento. Cada posición del flujo de salida es una lista de tuplas C#N, donde N es el número de veces que C aparece en el flujo de entrada.
  - a. Que sucede si uno de los clientes deja de enviar caracteres?  
**Respuesta:** No hay ningún problema dado que la función de mezcla ignora los nil
  - b. Puede extender la solución a N clientes?  
**Respuesta:** Si se puede extender a n-clientes ya que se puede mezclar un número indeterminado de streams
5. Explique por qué es necesario utilizar memoria intermedia limitada.  
**Respuesta:** Se requiere dado que los consumidores no son capaces de trabajar al mismo ritmo de los productores por lo tanto puede haber un desbordamiento de memoria y permitir una sincronización entre productores y consumidores.
6. Se declaran métodos auxiliares al crear la suma completa con el objetivo de hacer uso de la concurrencia

7. Se crea una función wait que trabaja sobre el flujo y permite esperar que se vayan ligando las variables requeridas.
8. Se crea una lista infinita de datos, pero esta no produce desbordamiento dado que solo se limita en tiempo de ejecución gracias a la carga perezosa.
9. Ejecute :

```
declare
fun lazy {HagaX} {Browse x} {Delay 3000} 1 end
fun lazy {HagaY} {Browse y} {Delay 6000} 2 end
fun lazy {HagaZ} {Browse z} {Delay 9000} 3 end
X={HagaX}
Y={HagaY}
Z={HagaZ}
{Browse (X+Y)+Z}
```

- a. Explique este comportamiento.

**Respuesta:** Se tienen 3 funciones perezosas que devuelven x y z, que son 3 valores enteros (1 2 3) y que se suman al final. Se activa la función perezosa realizando la asignación a la variable X, por lo que comienza la ejecución mostrando el carácter 'x' en la consola y esperando 3 segundos para devolver el valor 1 que se enlaza a la variable definida. Igualmente se realiza la asignación para las variables Y y Z, pero esta vez esperando 3 segundos y devolviendo 2 y esperando 9 segundos y devolviendo 3, respectivamente; para finalizar realizando la suma de los 3 valores.

- b. ¿Qué pasa si reemplazamos  $(X+Y)+Z$  por  $X+(Y+Z)$ ?

**Respuesta:** Por la prioridad matemática de los paréntesis se evalúa primero  $(Y + Z)$

The screenshot shows the Oz IDE with several files open: sum\_impares.oz, primosPereza.oz, terminacion.oz, contador.oz, foobar.oz, and test.oz. The test.oz file is active and contains the following code:

```

1
2 declare
3 fun lazy {HagaX} {Browse x} {Delay 3000} 1 end
4 fun lazy {HagaY} {Browse y} {Delay 6000} 2 end
5 fun lazy {HagaZ} {Browse z} {Delay 9000} 3 end
6 X={HagaX}
7 Y={HagaY}
8 Z={HagaZ}
9 % {Browse (X+Y)+Z}
10 {Browse X + (Y + Z)}

```

The Oz Browser window is open, showing the output of the code. The output is as follows:

| Browser | Selection | Options |
|---------|-----------|---------|
| y       |           |         |
| z       |           |         |
| x       |           |         |
| 6       |           |         |

- c. ¿Qué pasa si reemplazamos  $(X+Y)+Z$  por `thread X+Y end + Z`?

**Respuesta:** Se comporta de la misma manera que la versión inicial

The screenshot shows the Oz IDE with the same files as before. The test.oz file is active and contains the following code:

```

1
2 declare
3 fun lazy {HagaX} {Browse x} {Delay 3000} 1 end
4 fun lazy {HagaY} {Browse y} {Delay 6000} 2 end
5 fun lazy {HagaZ} {Browse z} {Delay 9000} 3 end
6 X={HagaX}
7 Y={HagaY}
8 Z={HagaZ}
9 % {Browse (X+Y)+Z}
10 % {Browse X + (Y + Z)}
11 {Browse thread X+Y end + Z}

```

The Oz Browser window is open, showing the output of the code. The output is as follows:

| Browser | Selection | Options |
|---------|-----------|---------|
| x       |           |         |
| y       |           |         |
| z       |           |         |
| 6       |           |         |

- d. ¿En cuál forma se presenta más rápido el resultado final?

**Respuesta:** Teniendo `thread X+Y end + Z` el tiempo es 18 segundos

```

test.oz
24 fun lazy {HagaZ}
25 local T1 T2 in
26 T1 = {Time.time}
27 {Browse z}
28 {Delay 9000}
29 T2 = {Time.time}
30 {Browse 'Tiempo HagaZ:' # (T2 - T1)}
31 3
32 end
33 end

34 local TStart TEnd in
35 TStart = {Time.time}
36 X = {HagaX}
37 Y = {HagaY}
38 Z = {HagaZ}
39
40 % {Browse (X+Y)+Z}
41 % {Browse X + (Y + Z)}
42 local TResult in
43 TResult = thread X+Y end + Z
44 TEnd = {Time.time}
45 {Browse 'Tiempo Total:' # (TEnd - TStart)}
46 {Browse TResult}
47 end
48 end
49 end

```

Oz Browser

| Browser         | Selection | Options |
|-----------------|-----------|---------|
| x               |           |         |
| 'Tiempo HagaX:' | #3        |         |
| y               |           |         |
| 'Tiempo HagaY:' | #6        |         |
| z               |           |         |
| 'Tiempo HagaZ:' | #9        |         |
| 'Tiempo Total:' | #18       |         |
| 6               |           |         |

Teniendo  $X + (Y + Z)$  el tiempo es 18 segundos

```

24 fun lazy {HagaZ}
25 local T1 T2 in
26 T1 = {Time.time}
27 {Browse z}
28 {Delay 9000}
29 T2 = {Time.time}
30 {Browse 'Tiempo HagaZ:' # (T2 - T1)}
31 3
32 end
33 end

34 local TStart TEnd in
35 TStart = {Time.time}
36 X = {HagaX}
37 Y = {HagaY}
38 Z = {HagaZ}
39
40 % {Browse (X+Y)+Z}
41 % {Browse X + (Y + Z)}
42 local TResult in
43 TResult = X + (Y + Z)
44 TEnd = {Time.time}
45 {Browse 'Tiempo Total:' # (TEnd - TStart)}
46 {Browse TResult}
47 end
48 end
49 end

```

Oz

| Browser         | Selection | Options |
|-----------------|-----------|---------|
| y               |           |         |
| 'Tiempo HagaY:' | #6        |         |
| z               |           |         |
| 'Tiempo HagaZ:' | #9        |         |
| x               |           |         |
| 'Tiempo HagaX:' | #3        |         |
| 'Tiempo Total:' | #18       |         |
| 6               |           |         |

Teniendo  $(X+Y)+Z$  el tiempo es 18 segundos

```

test.oz
24 fun lazy {HagaZ}
25 local T1 T2 in
26 T1 = {Time.time}
27 {Browse z}
28 {Delay 9000}
29 T2 = {Time.time}
30 {Browse 'Tiempo HagaZ:' # (T2 - T1)}
31 3
32 end
33 end
34
35 local TStart TEnd in
36 TStart = {Time.time}
37 X = {HagaX}
38 Y = {HagaY}
39 Z = {HagaZ}
40
41 % {Browse (X+Y)+Z}
42 % {Browse X + (Y + Z)}
43 local TResult in
44 TResult = (X+Y)+Z
45 TEnd = {Time.time}
46 {Browse 'Tiempo Total:' # (TEnd - TStart)}
47 {Browse TResult}
48 end

```

Oz B

Browser Selection

x  
'Tiempo HagaX:'#3

y  
'Tiempo HagaY:'#6

z  
'Tiempo HagaZ:'#9

'Tiempo Total:'#18

6

El tiempo en los 3 casos es el mismo dado que esta dado por el Delay y la suma de ellos da 18.

- e. ¿Cómo programaría la suma de  $n$  números enteros  $i_1, \dots, i_n$ , sabiendo que el entero  $i_j$  estara disponible sólo después de  $t_j$  ms, de manera que el resultado final se produzca lo más rápido posible?

**Respuesta:** Crear hilos para cada número  $i_j$  al producir su valor y utilizar un wait para que recolecte los valores lo más pronto posible.