Práctica 5

1. Teniendo en cuenta el ejemplo Deterministic Append, ¿qué sucede con la llamada Append X [3] [1 2 3], la cual lógicamente debería devolver X=[1 2]?

2. Considere las siguientes funciones:

```
fun {Digit}
        choice 0 [] 1 end
end
fun {TwoDigit}
        10*{Digit}+{Digit}
end
fun {StrangeTwoDigit}
        {Digit}+10*{Digit}
end
```

   ¿Qué muestran las siguientes sentencias:
   (a) {Browse {SolveAll Digit}}
   (b) {Browse {SolveAll TwoDigit}}
   (c) {Browse {SolveAll StrangeTwoDigit}}
   (d) Utilizando Digit generar todos los números palíndromos de 4 dígitos.

3. Dada las siguientes premisas:
   1. arc(a, b).
   2. arc(a, c).
   3. arc(a, f).
   4. arc(c, b).
   5. arc(b, d).
   6. arc(X, Y) → path(X, Y)
   7. arc(X, Y) ∧ path(Y, Z) → path(X, Z)

   Verificar mediante el método del absurdo la validez de ?¬(path(a, A)).
   Muestre claramente las sustituciones en cada paso. En los casos de éxitos mostrar cuál es la ligadura de A.

4. Dada las siguientes premisas :
   1. hijo(luisPerez, josePerez).
   2. hijo(mariaPerez, josePerez).
   3. hijo(miguelPerez, josePerez).
   4. hijo(luisPerez, lolaPerez).
   5. hombre(lolaPerez).
   6. hijo(X, Y) ∧ hombre(Y) → padre(Y, X)

   Verificar mediante el método del absurdo la validez de ?¬padre(X, Y)).
   Muestre claramente las sustituciones en cada paso. En los casos de éxitos mostrar cuál es la ligadura de X e Y.

5. Definir en Prolog los siguientes predicados:
    (a) pertenece(L, L1).
        L está en la lista L1.
    (b) length(L, N).
        Donde N es la longitud de la lista L.
    (c) elimina(X, Y S, ZS).
        ZS es la lista resultante de eliminar X de Y S .
    (d) Intersección de conjuntos.

6. Definir la relación MultipliacaLista(L1, F, L2) de forma que la lista L2 la lista resultado de multiplicar cada elemento de L1 por F
   Ejemplo de uso:
   ?- MultiplicaLista([3,2,5], 2, [6,4,10]).
   yes;
   ?- MultiplicaLista(X, 3, [6,9,3]).
   X = [2,3,1]
   Yes;

7. Definir la relación sub_set(L1 , L2), donde L2 es un subconjunto de L1 .
   No puede utilizar la relación pertenece(X, L).
   Ejemplo:
   ?- sub_set([1 , 2, 3],L).
   L = [1,2,3] ? ;
   L = [1,2] ? ;
   L = [1,3] ? ;
   L = [1] ? ;
   L = [2,3] ? ;
   L = [2] ? ;
   L = [3] ? ;
   L = []
   yes

8. Dado el siguiente programa Prolog:
        max(X,Y,Y) :- X =< Y.
        max(X,Y,X) :- X>Y.
   (a) ¿Es ineficiente? ¿Por qué?
   (b) Proponer una mejora de su eficiencia.
   (c) Construir el árbol SLD de ambas versiones con la siguiente consulta:
        ?max(2, 2, Z).

9. Dar un ejemplo del problema de la negación en Prolog.

10. Considerando el siguiente programa, ¿qué devuelven las siguientes consultas? Justificar en cada caso.

```
hombre(juan).
hombre(pedro).
mujer(X) :- not(hombre(X)).
```

(a) ?mujer(juan).
(b) ?mujer(X).
(c) ?not(not(hombre(X))).
(d) not(not(hombre(X))) es equivalente a hombre(X)?

11. Teniendo en cuenta el predicado pertenece(L, L1) del ejercicio 5 en su versión eficiente, usando el cut(!), y en su versión ineficiente, sin uso del cut(!), construir el correspondiente árbol SLD para la siguiente consulta: ?pertenece(X, [1, 2, 1, 3]).

Ejercicios Adicionales

12. Definir en Prolog los siguientes predicados:
(a) length(L, N). Donde N es la longitud de la lista L.
(b) append(A, B, C).
Verifica si C es la lista obtenida concatenando los elementos de la lista B a continuación de los elementos de la lista A.
(c) unión y diferencia de conjuntos.

Práctico 6

1. Describiendo los cambios que sufren tanto la memoria inmutable como la mutable, decir que muestra cada uno de los siguientes algoritmos:

(a)
```
local X Y in
    X=1
    {NewCell X Y}
    {Browse @Y}
end
```

(b)
```
local Z X C1 C2 Temp1 Temp2 in
    X=10
    {NewCell X C1}
    {NewCell C1 C2}
    {Browse @C1}
    {Exchange C1 Temp1 C2}
    {Exchange C2 Temp2 Temp1}
    {Browse @C2}
    {Browse @C1}
end
```

(c)
```
local X Y Z in
    X=1
    {NewCell X Y}
    {NewCell 1 Z}
    {Browse @Y==@Z}
end
```

2. Definir las siguientes expresiones en base a la operación primitiva Exchange.

(a) X = C := Y
(b) C := X
(c) X = @C

3. Traducir a lenguaje núcleo y ejecutar según la máquina abstracta.

```
local X C Y in
    Y=1
    {NewCell 0 C}
    {Exchange C X Y}
    {Browse @C}
    {Browse X}
end
```

4. Implementar la función Reverse L, la cual retorna la reversa de una lista utilizando estado.

5. Implementar en el lenguaje kernel con estado el tipo de datos Stack. Escribir un ejemplo de su uso.

6. Determinar el tipo de pasaje de parámetros que usan los siguientes lenguajes:
    (a) Pascal
    (b) C
    (c) C++
    (d) Java
    (e) Haskell

Ejercicios Adicionales

7. Dar esquemas en el lenguaje kernel con estado para simular cada uno de los mecanismos de pasajes de parámetros descriptos.

8. Describiendo los cambios que sufren tanto la memoria inmutable como la mutable, decir que muestra cada uno de los siguientes algoritmos:

```
(a) local X Y Z in
        {NewCell 2 Z}
        {NewCell Z X}
        {NewCell X Y}
        {Browse @Y}
        {Browse @X}
end
(b) local X Y Z in
        X=1
        {NewCell X Y}
        Z=Y
        {Browse Y==Z}
end
```

9. Traducir a lenguaje núcleo y ejecutar según la máquina abstracta.

```
local X C in
    X=1
    C={NewCell X}
    {Exchange C @C @C+1}
    {Browse @C}
    {Browse X}
end
```

Practica 7

Diseñar un algoritmo utilizando el lenguaje C para cada uno de los siguientes problemas.

1. Leer un carácter y dos números enteros. Si el carácter leído es un operador aritmético calcular la operación correspondiente, si es cualquier otro mostrar error. Hacer el programa utilizando la instrucción switch().

2. Leer un entero positivo y averiguar si es perfecto. Un número es perfecto cuando es igual a la suma de sus divisores excepto él mismo.

3. Leer por teclado un número entero largo e indicar si el número leído es o no capicúa. Para ello debe utilizarse un array unidimensional para almacenar cada una de las cifras del número leído. Se deben implementar dos funciones, una para descomponer el número en cifras y cargar el array, y otra para comparar las posiciones del array y determinar si el número es capicúa.

4. Calcular las raíces de una función utilizando el Método de la Bisección.

5. Cargar un array bidimensional de p * q y devolver un puntero apuntando a la que más suma.

6. Cargar un array bidimensional de p * q y devolver un arreglo A de longitud p donde A[i] contiene la suma de la fila i de la matriz. Puede utilizar únicamente punteros tanto para recorrer la matriz como para cargar los resultados en el arreglo.

Practica 8

1. Determinar en Pascal un experimento para ver la representación de la variable s:

```
Type DiaSemana = (Dom,Lun,Mar,Mie,Jue,Vie,Sab);
Var s:Set of DiaSemana;
```

2. La siguiente expresión calcula la dirección base del elemento i-ésimo de un vector (a lo C) T A[N], asumiendo que el tipo índice es el subrango [0..N-1].

$$dir(i) = dir\_base(A) + i * sizeof(T)$$

donde sizeof(T) es el tamaño de la representación de un valor de tipo T.

- Dar la expresión para una declaración de un vector a lo Pascal: A[li..ls] of T.
- Idem al anterior pero para una matriz de la forma A[li0..ls0,li0..ls0] of T.

3. Dado el siguiente programa Pascal, mostrar la pila de registros de activación hasta el punto *, mostrando los links estáticos y dinámicos.

```
Program P;
Var g:boolean;
    Function f2(Var b:boolean; y:integer):integer;
    begin
        if b then f2 := y + 1 else f2 := y - 1;
        b := not b
    end
    Procedure p1;
    Var x:integer;
        Function f1(y:integer):integer;
        begin
            f1 := y + f2(g,y)
        end
    begin {p1 body}
        x := 1
        x := f1(x) {*}
    end {p1}
begin
    g := True;
    p1
end.
```

4. Dado el siguiente programa Pascal, mostrar la pila de registros de activación hasta el punto *, mostrando los links estáticos y dinámicos.

```
Program P;
Var a:integer;
    Procedure P2(e:Char; Var f:integer; g:integer);
        Function fun(c:char; a:integer):integer;
        begin
            fun := ord(c) + a
        end;
    begin
        f := f + 1;
        g:= fun(e,f);
        writeln(g) {*}
    end;
    Procedure P1(b:integer);
    Var c:Char;
    begin
        c := chr(b);
        P2(c,b,a)
    end;
begin
    a := 64;
    P1(a);
end.
```

5. Dar un ejemplo en Pascal y en C que genere basura.

6. Dar un ejemplo en Pascal y en C el cual tenga una referencia colgada.

7. Diseñar un algoritmo en JAVA, que fuerce la ejecución del recolector de basuras. La creación de objetos debe detenerse en cuanto el recolector comience a funcionar. Se debe visualizar:

- la cantidad de objetos creados hasta la ejecución del recolector de basura.
- el orden en que los objetos van finalizando (ver método finalize()).
- un mensaje 'Todos los objetos finalizaron' cuando todos los objetos hayan finalizado.

8. Utilice la herramienta VisualVM [9] para observar el comportamiento en memoria del programa del ejercicio 7.

9. Garbage collector del lenguaje JAVA:
   - Analizar la estrategia por Generaciones que usa Java 11 (o posterior).
   - Estudie los parámetros de la MV Java que permite modificar (adaptar) el heap a las necesidades de una aplicación.
     Ejemplo: java -Xmx4g programaJava, indica que el Heap tendrá un tamaño máximo de 4Gb.
   - Utilizando los parámetros adecuados realice experimentos de tal forma que deje registros (.log) del comportamiento del GC. Chequear con un editor de textos el .log . También visualizar el log con la herramienta GcViewer [10]. Use los siguientes parámetros al lanzar el programa java:
     −verbose : gc −XX : +P rintGCDetails − XX : +U seConcM arkSweepGC − Xloggc :/directorio..../garbage.log ,
   - Usando el comando jmap[9] [11] realice un volcado de memoria de un programa Java en ejecución. Luego con la herramienta Eclipse Memory Analyzer [13] (u otra para los mismos fines como VisualVM) analice el uso de la memoria corriente.

---

[9] : jmap -dump:le=heap.hprof PID

---

10. Manejo de Memoria.
La figura 8.7 muestra un estado del HEAP de un proceso.
Mostrar gráficamente cómo queda el mismo luego de la ejecución del recolector de basura el cual se basa en la estrategia por copia.

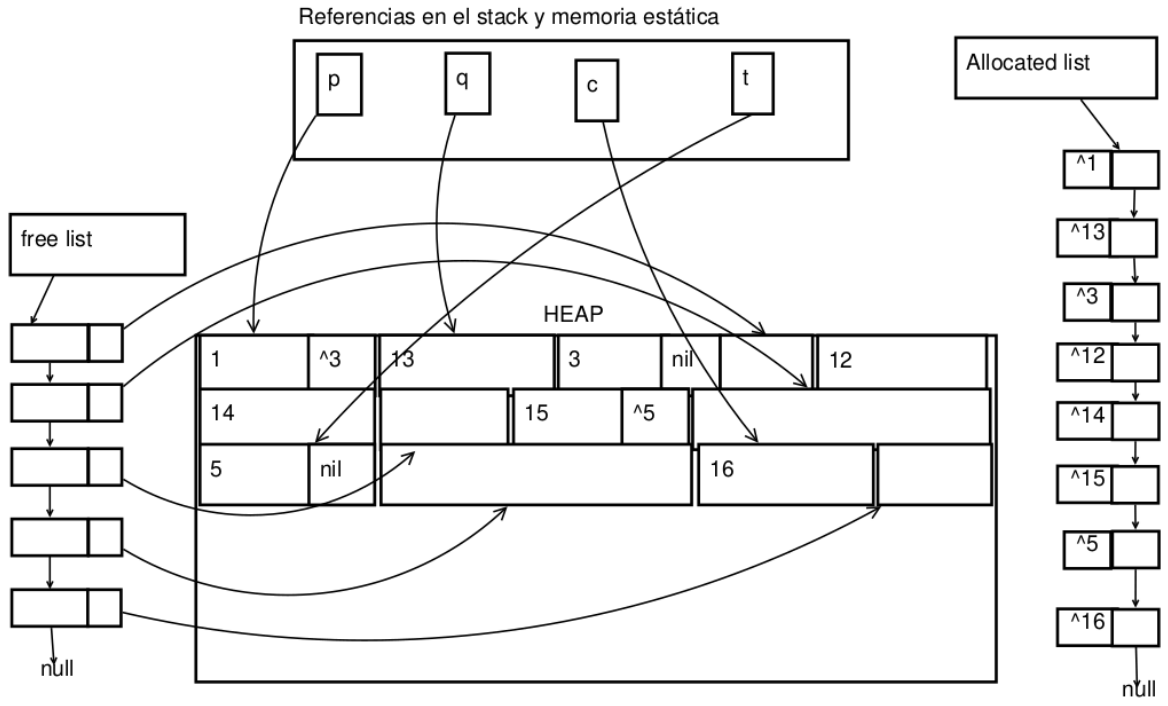


Fig. 8.7: Estado del Heap.

Ejercicios adicionales

11. Realizar un experimento en Pascal para determinar cómo se almacenan los valores de una matriz (arreglo bidimensional).

12. Escriba en Pascal o en C un programa que contenga una función recursiva factorial(n) y mostrar la pila de ejecución para n = 3.

Practica 9

1. Dada la siguiente clase en Oz, mostrar su uso extendiendo el programa con la creación de objetos y sus invocaciones a métodos. Compilar y ejecutar.

```
class Counter
        attr val
        meth init(V)
                val := V
        end
        meth inc(V)
                val := @val + V
        end
end
```

2. En lenguajes como C++ o Java, podemos hablar de mensajes o son en realidad invocaciones a procedimientos? Justificar la respuesta.

3. Describa las ventajas y desventajas de manejar las invocaciones a métodos como mensajes versus invocaciones a procedimientos.

4. Mostrar con un ejemplo que en Java utiliza dynamic dispatching y C++ utiliza static dispatching por default. Como hacer en C++ para que un método trabaje con dynamic dispatching.

5. Determinar si el destino (target) de la siguiente expresión puede ser determinada estáticamente o no:

```
…
super.m();
…
```

Justificar la respuesta.

6. Implementar en Java y C++ el TAD lista (heterogénea). Definir una clase abstracta y al menos dos implementaciones: sobre arreglos y nodos enlazados. Mostrar ejemplos de uso.

7. Explicar cómo se puede resolver el problema del rombo (con herencia múltiple) en C++ y Eiffel. Haga un experimento en C++.

8. Dado el siguiente programa, indicar que muestra por la salida estándar en cada uno de los casos. Corroborar mediante la ejecución del mismo.

```
public class B extends A2 {
        String m1 (B x) { return "BA2" ; }
        String m2 (B x) { return "Paso por B - m2(B) "+ this.m1(x) ; }
        String m2 (A2 x) { return "Paso por B - m2(A2) "+ this.m1(this) ; }
        public static void main(String[] args) {
                A2 a = new A2 ();
                A2 b2 = new B ();
                B b3 = new B();
                System.out.println (a.m2 (b3));         (1)
                System.out.println (b2.m2 (b2));        (2)
                System.out.println (b3.m2 ((B)b2));     (3)
                System.out.println (b3.m2 (a));         (4)
        }
}
class A2 {
        String m1 (A2 x) { return "AA2" ; }
        String m2 (A2 x) { return "Paso por A - m2(A2) "+ this.m1(x) ; }
}
```

9. Dado el siguiente programa en java, indicar que muestra por salida estándar en cada uno de los casos.

```
class A{
	public void type(){
		System.out.println("Soy A");
	}
	public void type(A _a) {
		_a.type();
	}
}
class B extends A{
	public void type(){
		System.out.println("Soy B");
	}
}
class C extends B{
	public void type() {
		System.out.println("Soy C");
	}
	public void type(B _b) {
		_b.type();
	}
}
class Test{
	public static void main(String[] args){
		A a = new A();
		B b= new B();
		B c = new C();
	}
	a.type(b);
	b.type(a);
	b.type(b);
	c.type((A) c);
	c.type((C) c);
}
```

10. Dé un ejemplo de una función en Java con número de parámetros variable.

11. Introspección.
Utilizando las funciones del módulo inspect, experimente en Python cómo verificar si un argumento es una clase, si un argumento es un método de una clase, mostrar la jerarquía de clases dada una clase, etc.

12. Escribir un programa en C++ que ordene los argumentos que pueda recibir desde la línea de comandos durante su invocación. Deberá usar los algoritmos de la STL (Standard Template Library) sort y alguno de los objetos-funciones (objetos que tienen sobrecargado el operador ()) de comparación (greater-equal, less, etc).

Ejercicios Adicionales

13. Modificar la clase lista de C++ para almacenar valores de un tipo T concreto (no de sus derivados).

14. Implementar en Java el TAD Pila de un tipo T. Muestra un ejemplo de uso.

15. Dar un ejemplo de cómo se puede implementar un diseño con herencia múltiple en Eiffel.

16. Escribir en C++ una clase parametrizada bintree<T>, la cual contenga las operaciones habituales de manejo de árboles binarios y que defina el iterador preorder-iterator, el cual debe permitir realizar un recorrido preorden del árbol.

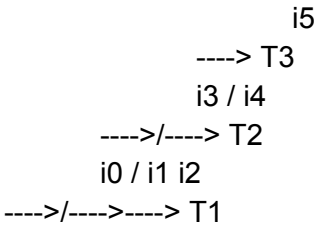
Un iterador es una abstracción de un puntero y debe mínimamente denir los siguientes operadores:

- operator==(): operador de comparación (igualdad).
- operator++(): operador de post-incremento. operator*(): operador de referenciación de un elemento del contenedor.

17. Escribir un programa que sume y multiplique los elementos de un vector. Los elementos deberán estar almacenados en un objeto de tipo vector<int>. Deberá utilizar el algoritmo genérico accumulate.

Practica 10

1. Dado el siguiente esquema de ejecución de threads:



   A. Dar el orden causal como un conjunto por extensión (orden parcial de ejecución de las instrucciones).
   B. Dar todos las posibles trazas de ejecución dadas por el interleaving.

2. Ejecutar manualmente, siguiendo la semántica del lenguaje concurrente declarativo del siguiente programa:

```
local X,Y in
        X = 1
        thread fun {$}
                Y = X
                X+1
        end
        end
        thread fun {$} Y end end
        {Browse X Y}
end
```

3. Implementar en Oz un programa con un thread que genere la sucesión de los números de Fibonacci (hasta un cierto N) y otro thread que los vaya mostrando por salida estándar (Browse). Este problema es una instancia del productor-consumidor donde la lista es un stream.

4. Dado el siguiente programa C

```
#include <stdio.h>
#include <unistd.h>/* por getpid() */
#define N (2000)
int main(void)
{
        FILE * f ;
        int i, value;
        for (i=0; i<N; i++) {
                f = fopen("seqno.txt","r+");
                fscanf(f,"%d", &value);
                value++;
                rewind(f);
                fprintf(f,"%6d\n", value);
                fflush(f);
                printf("Process id: %d, value: %d\n", getpid(), value);
                fflush(stdout); /* forzar la escritura al archivo */
                fclose(f);
                sleep(2);
        }
}
```

A. Crear el archivo seqno.txt conteniendo una única línea con un cero.
B. Correr dos (o más) instancias del programa en forma secuencial (ej: prog ; prog) y verificar que la salida es la esperada.
C. Correr dos (o más) instancias del programa en forma concurrente (ej: prog & prog) y verificar el resultado final en el archivo.
D. Existe una race condition ? Justificar.

5. Modificar el programa anterior para que logre su objetivo cuando se ejecuta en forma concurrente.
Ayuda: usar la función de biblioteca lockf().

6. Dado el siguiente programa.

```
local Y1 Y2 C in
        C={NewCell nil}
        thread X=1 C:=X|@C Y1='listo' end
        thread C:=2|@C Y2='listo' end
        {Barrier Y1 Y2}
        {Browse @C}
end
```

- Defina una traza de ejecución por la cual la lista correspondiente a la Celda C termina con un solo elemento.
- ¿Cuáles son los posibles valores que muestra {Browse @C}?
- Usando algún mecanismo de sincronización, sincronice la región crítica para que la lista siempre termine con 2 elementos.

7. Implementar en Java una clase Semaphore el cual se comporte como los semáforos definidos en este capítulo.

8. Considere la siguiente clase JAVA. Escriba un programa que genere dos o más thread que actúen sobre un objeto de dicha clase.

```
public class Cuenta {
        private int saldo;
        public Cuenta(int saldoInicial) {
                saldo = saldoInicial;
        }
        public void deposita(int monto) {
                saldo += monto;
        }
        public void retira(int monto) {
                saldo -= monto;
        }
        public int getSaldo() { return saldo; }
}
```

(a) Verificar si existen condiciones de carrera.
(b) ¿Qué mecanismos ofrece Java para sincronizar los procesos?
(c) Utilice los semáforos definidos en el ejercicio 7 para garantizar la exclusión mutua a los métodos de la clase Cuenta.

9. El problema de los filósofos comensales es un problema clásico para el estudio de deadlocks. Existen n filósofos, los cuales, repetitivamente, piensan por un momento y luego toman dos tenedores que se encuentran al lado del plato y comen, para finalmente dejar los tenedores nuevamente en la mesa.
Cada filósofo tiene un plato y hay n tenedores: uno entre cada plato.

(a) Describir un estado en que puede ocurrir deadlock.
(b) Escribir un programa que modele el problema libre de deadlock.

10. Implementar y ejecutar en Erlang el módulo areaserver definido en el capítulo.

11. Analizar cómo soporta Erlang (en caso de que lo cubra) los siguientes conceptos:

(a) Paradigma.
(b) Sistema de Tipos.
(c) Polimorfismo.
(d) Modelo Concurrente. Comunicación sincrónica o asincrónica.
(e) Técnicas de Alto Orden.
(f) Ejemplos de aplicaciones implementadas parcial o totalmente en Erlang.

Ejercicios Adicionales

12. Implementar un programa concurrente en Oz con estado que contenga al menos dos threads realizando acciones sobre una instancia de la clase Stack definida en este capítulo. Utilizar locks para atomizar las operaciones sobre el stack.

13. Implementar en Oz un procedimiento {Wait X} el cual debe hacer esperar (bloquear) al thread que lo ejecuta hasta que X se ligue.

14. Implementar en Oz un procedimiento {Barrier [X1 X2 … Xn]} que se comporte como una barrera.

15. Dada la siguiente clase Java, implementar el problema del productor-consumidor (o bounded buffer). Definir las clases Producer y Consumer que representen threads produciendo y consumiendo valores a/desde una instancia de Buffer.

```
class Buffer {
        int[] buf;
        int first, last, n, i;
        public void init(int size) {
                buf=new int[size];
                n=size; i=0; first=0; last=0;
        }
        public synchronized void put(int x) {
                while (i<n) wait();
                buf[last]=x;
                last=(last+1)%n;
                i=i+1;
                notifyAll();
        }
        public synchronized int get() {
                int x;
                while (i==0) wait();
                x=buf[first];
                first=(first+1)%n;
                i=i-1;
                notifyAll();
                return x;
        }
}
```

16. Qué sucedería si en la clase Buffer del ejercicio anterior se reemplaza cada notifyAll() por notify()?

17. Modificar el programa Java del productor-consumidor usando los semáforos definidos anteriormente.

18. Un port object (o agente) es un objeto que combina múltiples puertos con un único stream. Esto permite la comunicación entre procesos de la forma muchos a uno. Los port objects son generalmente usados para modelar e implementar sistemas distribuidos. Un port object procesa los mensajes que arriban por sus puertos.
Implementar una aplicación del tipo múltiples escritores, un lector, el cual es una instancia del problema general de productores y consumidores, usando port objects.