**Resumen Taller:** Parcial Teórico- Práctico Promoción.

Modulo imperativo:

**Definiciones**

**vector/arreglo:** estructura de datos compuesta que permite acceder a cada componente por una variable índice, por eso se dice “indexada”. Posiciones contiguas de memoria. Homogénea, estática, acceso directo a los datos (indexada), lineal (a partir de un dato puede acceder al siguiente o anterior). DL y DF.

**Lista:** estructura de datos lineal, compuesta por nodos. Cada nodo posee un dato y la dirección del siguiente nodo. Se debe recorrer desde el primero (no indexada). No tiene posiciones contiguas de memoria necesariamente, es dinámica. New y dispose, generar espacio y eliminarlo. Homogénea, dinámica, acceso secuencial, lineal.

**Algoritmo de ordenación:** proceso por el cual un conjunto de elementos puede ser ordenado. Se tiene en cuenta para la elección: tiempo de ejecución, facilidad de escritura, memoria utilizada, complejidad de las estructuras, como actúa dependiendo los distintos tipos de desorden.

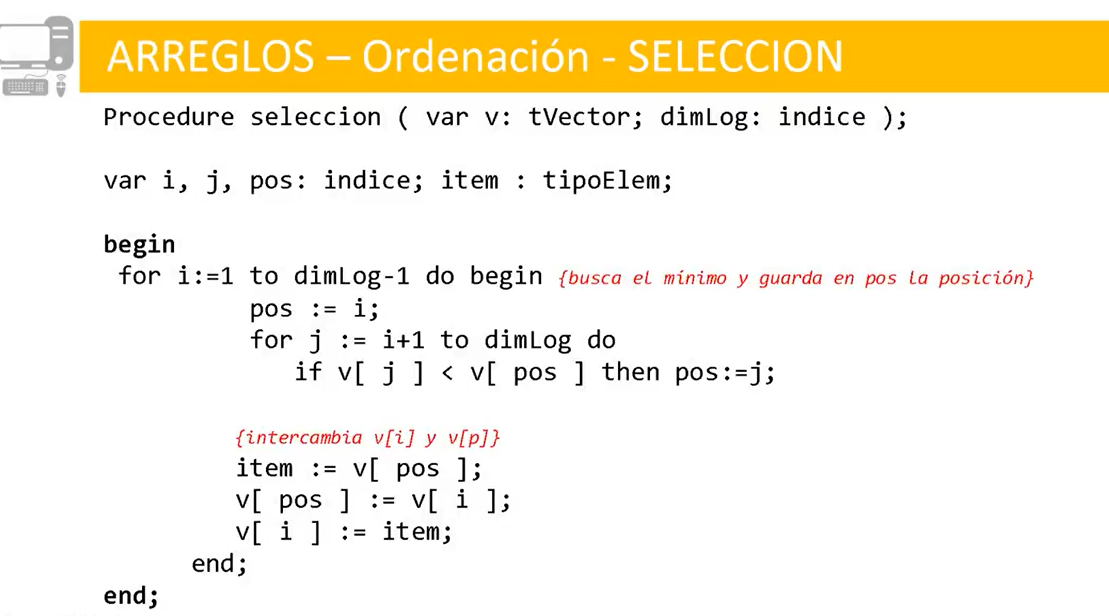
**Recursión:** técnica de resolución de problemas que consiste en dividir un problema en instancias más pequeñas hasta que se llega al caso base, que es un problema con solución directa o trivial. Las soluciones recursivas no son eficientes en cuanto al uso de memoria, debido a que en memoria se mantienen activos todos los llamados hasta que el ultimo llamado se termina y ahí se empieza a liberar.

**Arboles:** estructura no lineal, dinámica, homogénea, compuesta, que se utiliza cuando existe un tipo de jerarquía en el problema. Estructura de datos jerárquica (se sabe para cada nodo quien es su padre), formada por nodos, el nodo principal se denomina raíz, y los que no tienen hijos, hojas del árbol. Arboles binarios, como máximo cada nodo puede tener dos hijos. Arboles binarios de búsqueda, hay una condición que se respeta para ubicar los distintos datos. La búsqueda en un arbolBB es de orden logarítmico, por lo que es mucho más rápida que una lista (orden lineal).

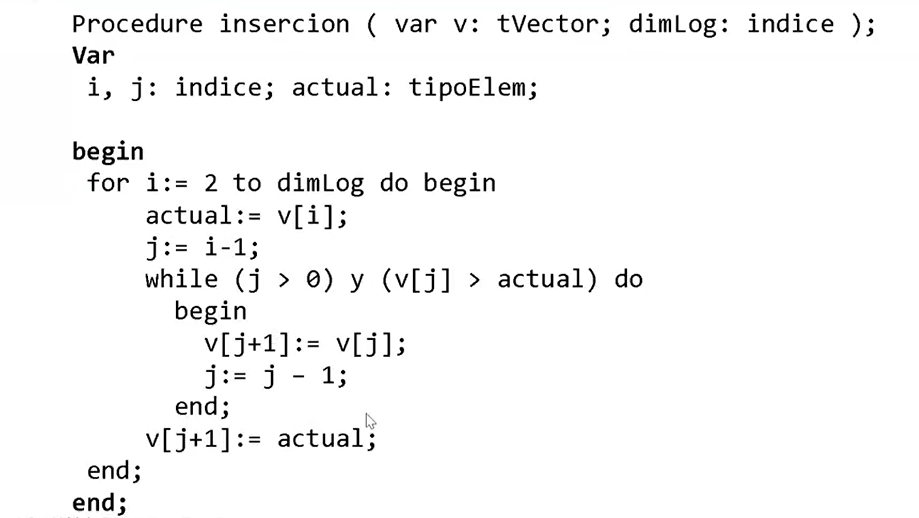
**Módulos**

**Algoritmos de ordenación en vectores:**

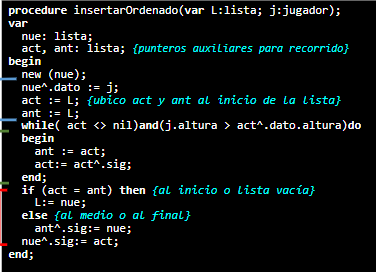
* **selección:** más simple, mayor tiempo. N vueltas, n= dimensión del arreglo-1. Tiempo de ejecución N², fácil escritura, no necesita estructura extra, si esta ordenado, desordenado o inverso el tiempo no varía. Encuentra el elemento más chico a partir de la posición actual, cada vuelta pos= pos +1.

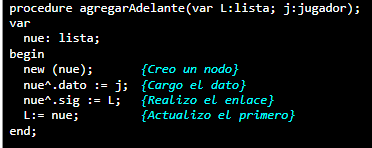
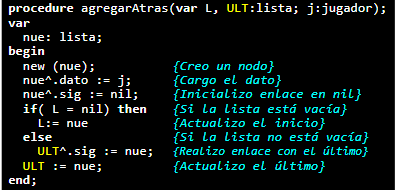


* **Inserción:** más complejo, menor tiempo. N vueltas, n= dimensión del arreglo. Por cada elemento busca en que posición debería insertarse. Si el elemento está en la pos correcta no realiza ningún movimiento. Tiempo de ejecución N², difícil escritura, no necesita estructura extra, si esta ordenado solo compara, tiempo N.



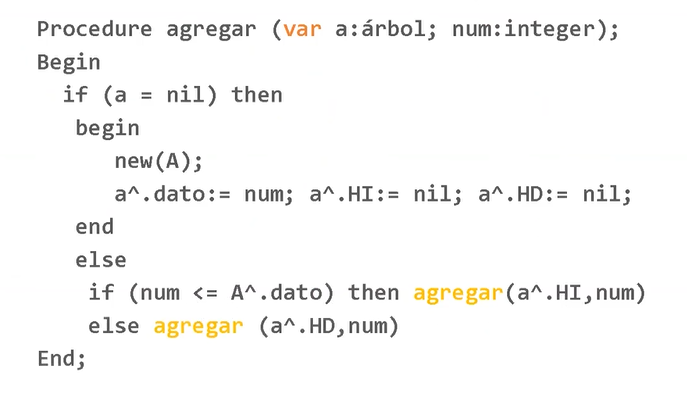
**Módulos listas:**

****

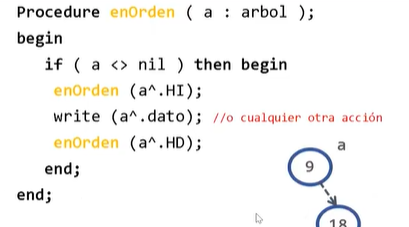
****

**Módulos arboles:**

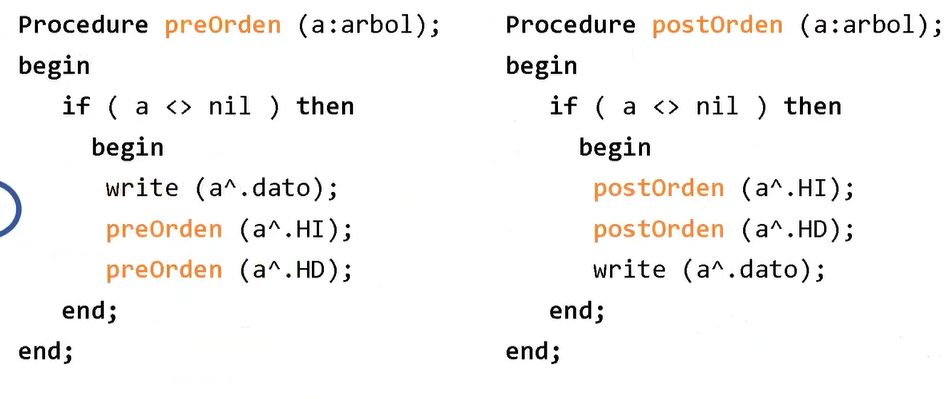
* Agregar en el árbol:



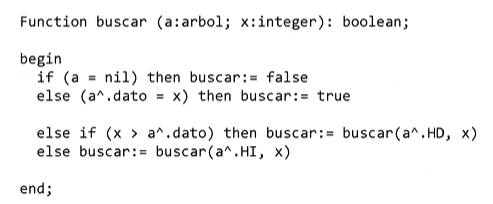
* recorrer: de menor a mayor.



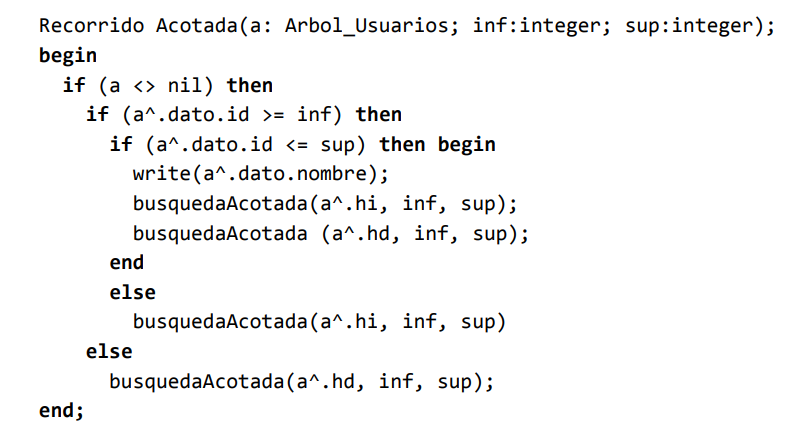
* Preorden y PostOrden:



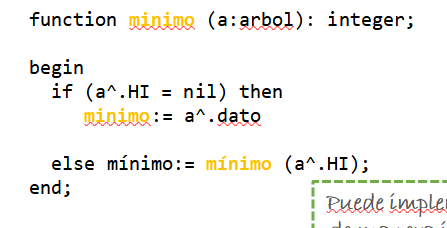
* buscar:



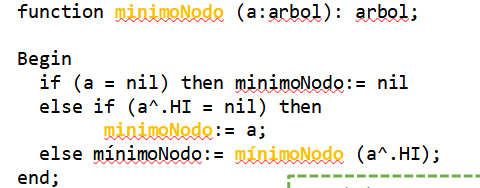
* buscar entre:



* valor mínimo: el que está más a la izquierda



* contenido mínimo: el nodo que contiene el dato menor



* lo mismo para los máximos, pero con HD.

Modulo objetos:

**Definiciones**

**casting:** transformación que hace java sobre los resultados y tipos de variables. Por ejemplo, si tengo int i = ¾ debería dar 1,3 pero como es una división entre enteros java interpreta que debe devolver un numero entero. Se debe aclarar double antes de la división.

El tamaño de los arreglos se determina en la ejecución no en la compilación como en pascal. Siempre enteros y arrancan en 0.

matrices: colección indexada y ordenada de elementos. Es una estructura de datos compuesta que permite acceder a cada componente usando dos índices que permiten ubicar el elemento. Homogénea, estática, indexada, lineal.

**Objeto:** es una abstracción de un objeto/de una entidad del mundo real. Abstraer es tomar los elementos que más nos importan de la entidad y asumir que esa entidad está representada solo por esas características, el resto no importan. Se define el estado interno (los valores de sus características/atributos, se implementa mediante variables de instancia) y el comportamiento (que acciones/servicios sabe realizar, se implementan mediante métodos de instancia que operan sobre el estado interno), los servicios que ofrece al exterior constituyen la interfaz.

**Encapsulamiento:** se oculta la implementación del objeto hacia el exterior, donde solo se conoce la interfaz del mismo. Facilita el mantenimiento y evolución del sistema ya que no hay dependencias entre las partes del mismo.

**Envío de mensaje:** provoca la ejecución del método indicado por el nombre del mensaje. Puede llevar datos (parámetros del método), y puede devolver datos (resultado del método).

**Clase:** describe un conjunto de objetos comunes (mismo tipo). Consta de variables de instancia que representan el estado del objeto y el código de los métodos que implementan estos objetos (públicos y privados).

La **instanciación** se realiza enviando un mensaje de creación a la clase. Reserva espacio para el objeto, ejecuta el código del constructor.

Los programas se organizan como una colección de objetos que cooperan entre sí enviándose mensajes. Cada objeto es instancia de una clase. Los objetos se crean a medida que se necesitan. Cuando los objetos ya no son necesarios (cuando el objeto ya no es referenciado) se borran de memoria. Los parámetros son únicamente por valor. Las variables originales no se ven modificadas por los métodos, pero las variables de instancia de los objetos si, solo no se ve modificado si se cambia al objeto donde se apunta.

**Creación de objetos:** Secuencia de pasos en la creación:

• Reserva de Memoria. Las variables de instancia se inicializan a valores por defecto o explícito (si hubiese).

• Ejecución del Constructor (código para inicializar variables de instancia con los valores que

enviamos en el mensaje de creación).

• Asignación de la referencia a la variable.

Si la clase no declara ningún constructor, Java incluye uno sin parámetros y sin

código (constructor nulo).

Puede haber varios constructores para la clase (sobrecarga).

**Herencia:** Es un mecanismo que permite que una clase herede características y

comportamiento (atributos y métodos) de otra clase (clase padre o súper clase). A su vez, la clase hija define sus propias características y comportamiento. Tiene como ventaja la reutilización del código.

**Superclase** es la primera clase de la jerarquía, la superclase contiene atributos y métodos que son comunes a todas las subclases, clases que sean herencia de la superclase.

El “**this”** sirve para cuando se tiene el mismo nombre de variables, para acceder a las variables del objeto actual y para cuando se requiere un metido de clases superiores a la actual para acceder a atributos heredados. Objeto que provoca la ejecución del método.

Una **clase abstracta** es una clase que no puede ser instanciada (no se pueden

crear objetos de esta clase). Define características y comportamiento común

para un conjunto de clases (subclases). Puede definir **métodos abstractos** (sin

implementación) que deben ser implementados por las subclases.

Uso “**super”** cuando quiero buscar un método de una superclase que también está definido en la clase actual.

**Encapsulamiento:** permite construir componentes autónomos de software, es decir independientes de los demás componentes. La independencia se logra ocultando detalles internos (implementación) de cada componente. Una vez encapsulado, el componente se puede ver como una caja negra de la cual sólo se conoce su interfaz.

Se ocultan los detalles internos, toda la implementación del objeto y las variables de instancia, y solo se accede a él mediante la interfaz.

**Polimorfismo:** capacidad de tener objetos de clases distintas que son capaces de responder a los mismos mensajes, de manera distinta.

**Binding dinámico:** se determina en tiempo de ejecución el método(código) a ejecutar para responder al mensaje. No en la compilación.

Beneficios de la programación orientada a objetos:

natural. Fiable. Reusable. Fácil de mantener.

Modulo concurrente:

**Definiciones**

Un **programa concurrente** se divide en tareas (2 o más), las cuales se ejecutan al mismo tiempo y realizan acciones para cumplir un objetivo común. Para esto pueden: compartir recursos, coordinarse y cooperar. Cualquier lenguaje que brinde concurrencia debe proveer mecanismos para comunicar y sincronizar procesos.

**Área compartida:** todos los robots acceden; **área parcialmente compartida:** dos o más acceden, pero no todos; **área privada:** solo uno accede. Área privada memoria local, área pc memoria que acceden dos o más procesadores, el área compartida la memoria principal.

**Comunicación:** los procesos involucrados realizan un pasaje de mensajes y comparten memoria:

* Pasaje de mensajes: Es necesario establecer un canal (lógico o físico) para transmitir información entre procesos. También el lenguaje debe proveer un protocolo adecuado. Para que la comunicación sea efectiva los procesos deben “saber” cuándo tienen mensajes para leer y cuando deben transmitir mensajes.
* Memoria compartida: Los procesos intercambian información sobre la memoria compartida o actúan coordinadamente sobre datos residentes en ella. Lógicamente no pueden operar simultáneamente sobre la memoria compartida, lo que obliga a bloquear y liberar el acceso a la memoria. La solución más elemental es una variable de control que habilite o no el acceso de un proceso a la memoria compartida.

Él **envió de mensajes** es a sincrónico, por lo que no debe esperar a que se dé la comunicación para seguir su ejecución, se puede enviar una variable o un dato inmediato de tipo entero o boolean y el nombre del robot al que se lo envía (no el tipo).

La **recepción de mensaje** es sincrónica, debe esperar a recibir el mensaje para seguir ejecutándose. Siempre se recibe en una variable (entero o boolean).

**Bloquear recurso:** Dado un recurso compartido (por 2 o más procesos) que está DISPONIBLE se bloquea ese recurso para que otro proceso no pueda accederlo. Puede realizarlo el programador o el Sistema Operativo. Hay que bloquear un recurso cuando puede ser accedido por dos o más procesos de un programa y se hace en el menor tiempo posible. Lo mismo para el **Liberar recurso**. En el entorno CMRE la liberación y bloqueo de recurso se hace por parte del programador. Todos los recursos deben quedar liberados al finalizad la ejecución.

**Tipos de problemas reales:**

* **sincronización por barrera:** Múltiples procesos se ejecutan concurrentemente, hasta que llegan a un punto especial, llamado barrera. Los procesos que llegan a la barrera deben detenerse y esperar que todos los procesos alcancen la barrera, luego podrán retomar su actividad (hasta finalizar o hasta alcanzar la próxima barrera).  
  Para esto los procesos deben avisar que llegaron.
* **passing the baton:** Sólo un proceso a la vez, el que posee el testigo (baton), se mantiene activo. Cuando el proceso activo completa su tarea, entrega el baton a otro proceso. El proceso que entregó el baton queda a la espera hasta recibirlo nuevamente. Para esto los procesos deben tener una forma de comunicarse con el otro proceso.
* **productor/consumidor:**   
  - Productores: trabajan para generar algún recurso y almacenarlo en un espacio compartido, deben avisar que dejaron el recurso.   
  - Consumidores: utilizan los recursos generados por los productores para realizar su trabajo.  
  Para esto los procesos deben coordinar donde almacenan los datos los productores, los consumidores consumen el recurso, deben enterarse que el recurso fue producido.
* **servidor/cliente:**   
  - Servidores: permanecen inactivos hasta que un cliente les solicita algo. Cuando reciben una solicitud, realizan su tarea, entregan el resultado y vuelven a “dormir”  
  - Clientes: realizan su trabajo de manera independiente, hasta que requieren algo de un servidor. Entonces realizan una solicitud a un proceso servidor, y esperan hasta que recibir la respuesta. Cuando esto sucede, el cliente continúa su trabajo.  
  Para esto los procesos cliente deben realizar sus pedidos y el servidor debe administrar como los atiende.
* **master/slave:**   
  -Maestro: deriva tareas a otros procesos (trabajadores)  
  - Esclavos: realizan la tarea solicitada y envían el resultado al jefe, quedando a la espera de la siguiente tarea  
  Para esto el proceso jefe determina cuantos trabajadores necesita, cómo les reparte la tarea, cómo recibe los resultados.