**ESTRUCTURAS DINÁMICAS DE DATOS**

**(Pilas, Colas y Listas Enlazadas)**

**INTRODUCCIÓN**

Los datos estudiados hasta ahora se denominan *estáticos*. Ello es debido a que las variables son direcciones simbólicas de posiciones de memoria; esta relación entre nombres de variables y posiciones de memoria es una *relación estática* que se establece por la declaración de las variables de una unidad de programa y que se establece durante la ejecución de esa unidad.

Aunque el contenido de una posición de memoria asociada con una variable puede cambiar durante la ejecución, es decir, el valor de la variable puede cambiar, las variables por sí mismas no se pueden crear ni destruir durante la ejecución. En consecuencia, las variables consideradas hasta este punto se denominan *variables estáticas*.

En algunas ocasiones, sin embargo, no se conoce por adelantado cuánta memoria se requerirá para un programa. En esos casos es conveniente disponer de un método para adquirir posiciones adicionales de memoria a medida que se necesiten durante la ejecución del programa y liberarlas cuando no se necesitan. Las variables que se crean y están disponibles durante la ejecución de un programa se llaman *variables dinámicas*. Estas variables se representan con un tipo de datos conocido como *puntero*. Las variables dinámicas se utilizan para crear *estructuras dinámicas de datos* que se pueden ampliar y comprimir a medida que se requieran durante la ejecución del programa. Una estructura de datos dinámica es una colección de elementos denominados nodos de la estructura —normalmente de tipo registro— que son enlazados juntos. Las estructuras dinámicas de datos se clasifican en

lineales y no lineales.

**ESTRUCTURAS DE DATOS**

Los tipos y estructuras de datos primitivos, tales como *enteros, real* y *carácter* son utilizados para construir tipos más complicados como arreglos (*arrays)* y *registros*, denominados estructuras de datos compuestos. Tienen una estructura porque sus datos están relacionados entre sí. Las estructuras compuestas, tales como *arreglos* y *registros*, están soportadas en la mayoría de los lenguajes de programación, debido a que son necesarias en casi todas las aplicaciones.

La potencia y flexibilidad de un lenguaje está directamente relacionada con las estructuras de datos que posee. La programación de algoritmos complicados puede resultar muy difícil en un lenguaje con estructuras de datos limitados, caso de Fortran y Cobol. En ese caso es conveniente pensar en la implementación con lenguajes que soporten apuntadores (punteros) como **C** y **C++**.

Cuando una aplicación particular requiere una estructura de datos no soportada por el lenguaje, se hace necesaria una labor de programación para representarla. Se dice que necesitamos *implementar* la estructura de datos. Esto naturalmente significa más trabajo para el programador.

Un método muy eficaz es diseñar procedimientos y funciones que ejecuten las operaciones realizadas por las estructuras de datos. Sin embargo, con las estructuras vistas hasta ahora, los arreglos, tienen dos inconvenientes:

1. *La reorganización de una lista, si ésta implica movimiento de muchos elementos de datos, puede ser muy costosa*
2. Son *estructuras de datos estáticas*.

Una estructura de datos se dice que es *estática* cuando el tamaño ocupado en memoria es fijo, es decir, siempre ocupa la misma cantidad de espacio en memoria. Por consiguiente, si se representa una lista como vector, se debe anticipar (*declarar* o *dimensionar*) la longitud de esa lista cuando se escribe un programa; es imposible ampliar el espacio de memoria disponible. En consecuencia, puede resultar difícil representar diferentes estructuras de datos.

Los arreglos unidimensionales son estructuras estáticas lineales ordenadas secuencialmente. *Las estructuras se convierten en dinámicas cuando los elementos pueden ser insertados o suprimidos directamente sin necesidad de algoritmos complejos.* Se distinguen las estructuras dinámicas de las estáticas por los modos en que se realizan las inserciones y borrados de elementos.

**Estructuras dinámicas de datos**

Las estructuras dinámicas de datos son estructuras que “crecen a medida que se ejecuta un programa”. Una *estructura dinámica de datos* es una colección de elementos —llamados *nodos*— que son normalmente registros. Al contrario que un arreglo, que contiene espacio para almacenar un número fijo de elementos, una estructura dinámica de datos se amplía y contrae durante la ejecución del programa, basada en los registros de almacenamiento de datos del programa.

Las estructuras dinámicas de datos se pueden dividir en dos grandes grupos:

pilas

*lineales* colas

listas enlazadas

árboles

*no lineales*

grafos

Las estructuras dinámicas de datos se utilizan para almacenamiento de datos del mundo real que están cambiando constantemente. Un ejemplo típico como estructura estática de datos sería: la lista de pasajeros de una línea aérea. Si esta lista se mantuviera en orden alfabético en un arreglo, sería necesario hacer espacio para insertar un nuevo pasajero por orden alfabético. Esto requiere utilizar un bucle para copiar los datos del registro de cada pasajero en el siguiente elemento del arreglo. Si en su lugar se utilizara una estructura dinámica de datos, los nuevos datos del pasajero se pueden insertar simplemente entre dos registros existentes sin un mínimo esfuerzo.

Las estructuras dinámicas de datos son extremadamente flexibles. Como se ha descrito anteriormente, es relativamente fácil añadir nueva información creando un nuevo nodo e insertándolo entre nodos existentes. Se verá que es también relativamente fácil modificar estructuras dinámicas de datos, eliminando o borrando un nodo existente.

Una *estructura estática de datos* es aquella cuya estructura se especifica en el momento en que se escribe el programa y no puede ser modificada por el programa. Los valores de sus diferentes elementos pueden variar, pero no su estructura, ya que ésta es fija. Una *estructura dinámica de datos* puede modificar su estructura mediante el programa. Puede ampliar o limitar su tamaño mientras se ejecuta el programa.

**LISTAS**

Una *lista lineal* es un conjunto de elementos de un tipo dado que pueden variar en número y donde cada elemento tiene un único predecesor y un único sucesor o siguiente, excepto el primero y el último de la lista. Esta es una definición muy general que incluye los archivos y vectores.

Los elementos de una lista lineal se almacenan normalmente contiguos —un elemento detrás de otro— en posiciones consecutivas de la memoria. Las sucesivas entradas en una guía o directorio telefónico, por ejemplo, están en líneas sucesivas, excepto en las partes superior e inferior de cada columna. Una lista lineal se almacena en la memoria principal de una computadora en posiciones sucesivas de memoria; cuando se almacenan en un disco magnético, los elementos sucesivos se presentan en sucesión en el disco. Esta asignación de memoria se denomina *almacenamiento*

*secuencial*. Existe otro tipo de almacenamiento denominado *encadenado* o *enlazado*.

Las líneas así definidas se denominan *contiguas*. Las operaciones que se pueden realizar con listas lineales contiguas son:

1. Insertar, eliminar o localizar un elemento.

2. Determinar el tamaño —número de elementos— de la lista.

3. Recorrer la lista para localizar un determinado elemento.

4. Clasificar los elementos de la lista en orden ascendente o descendente.

5. Unir dos o más listas en una sola.

6. Dividir una lista en varias sublistas.

7. Copiar la lista.

8. Borrar la lista.

Una lista lineal contigua se almacena en la memoria de la computadora en posiciones sucesivas o adyacentes y se procesa como un arreglo unidimensional. En este caso, el acceso a cualquier elemento de la lista y la adición de nuevos elementos es fácil; sin embargo, la inserción o borrado requiere un desplazamiento de lugar de los elementos que le siguen y, en consecuencia, el diseño de un algoritmo específico.

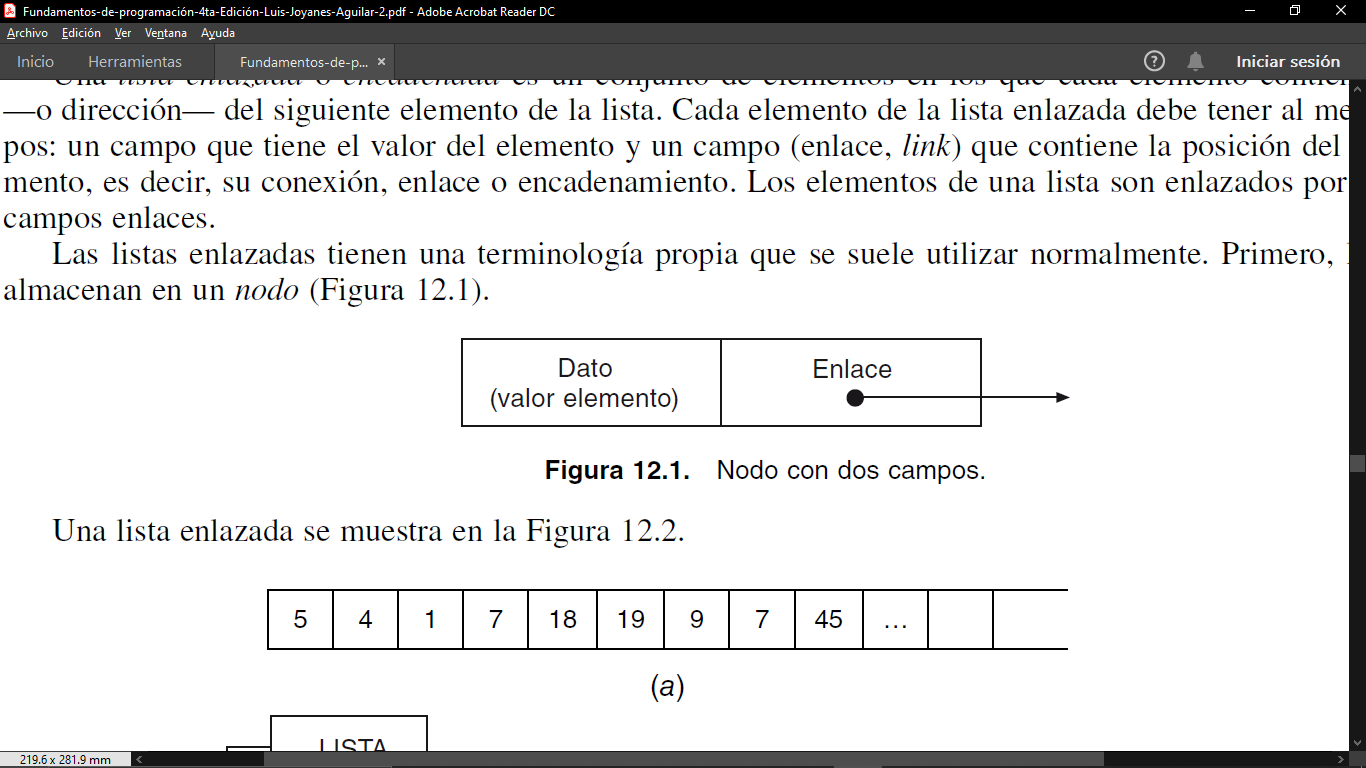
Para permitir operaciones con listas como arreglos se deben dimensionar éstos con tamaño suficiente para que contengan todos los posibles elementos de la lista.

**LISTAS ENLAZADAS**

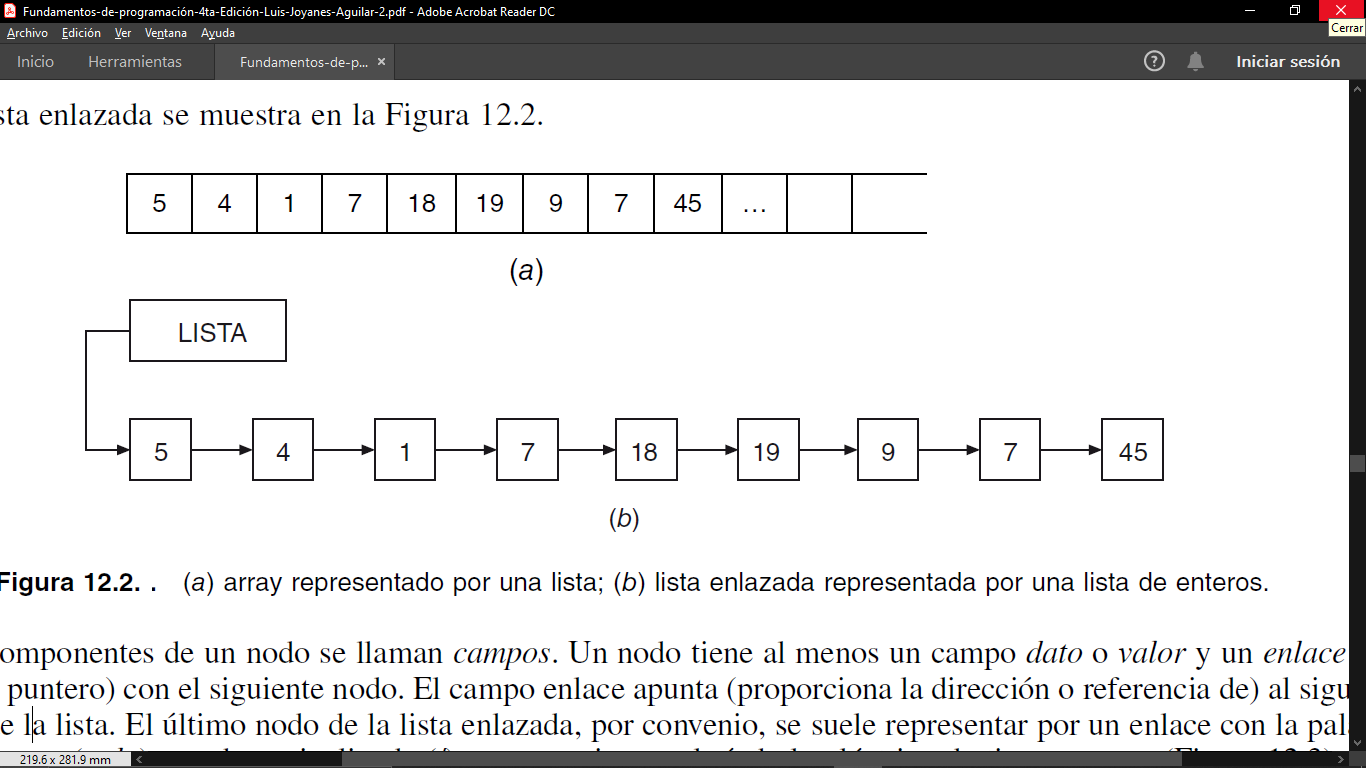
Los inconvenientes de las listas contiguas se eliminan con las listas enlazadas. Se pueden almacenar los elementos de una lista lineal en posiciones de memoria que no sean contiguas o adyacentes.

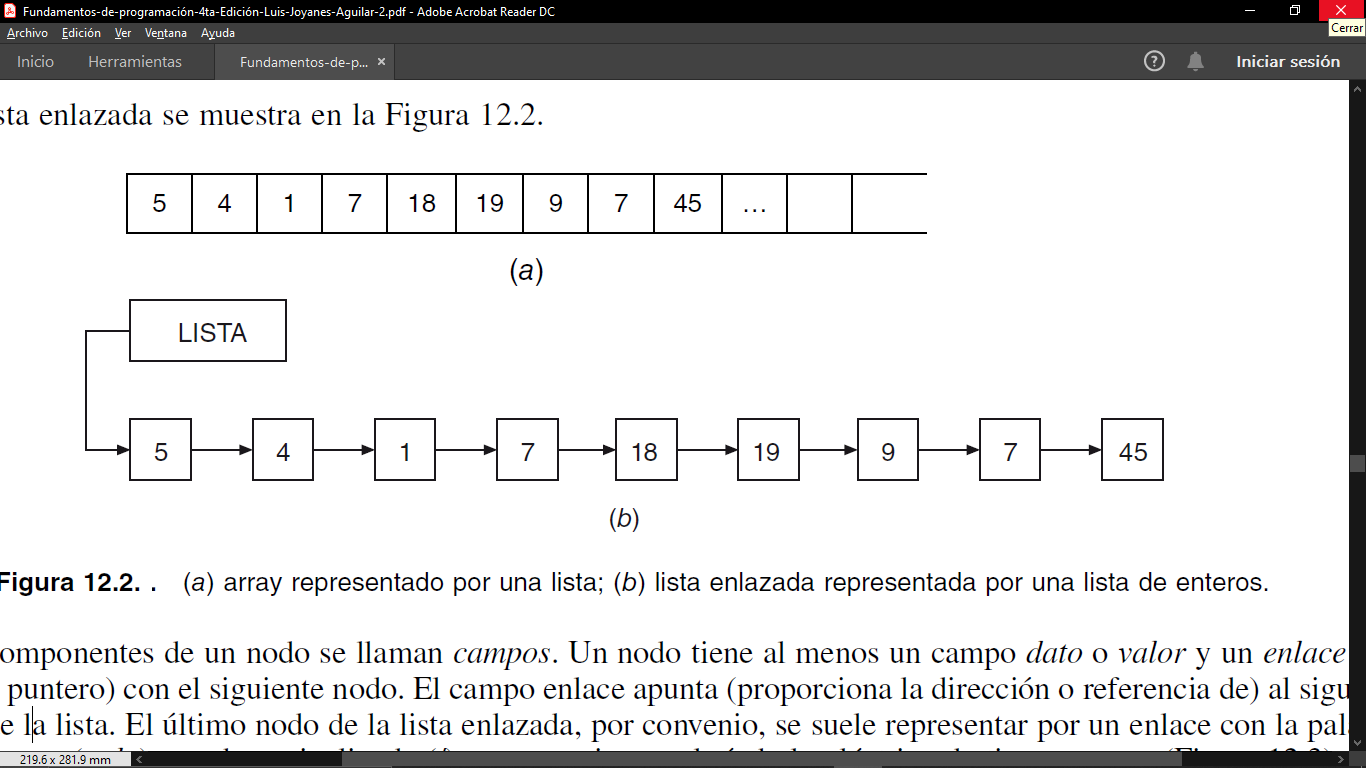
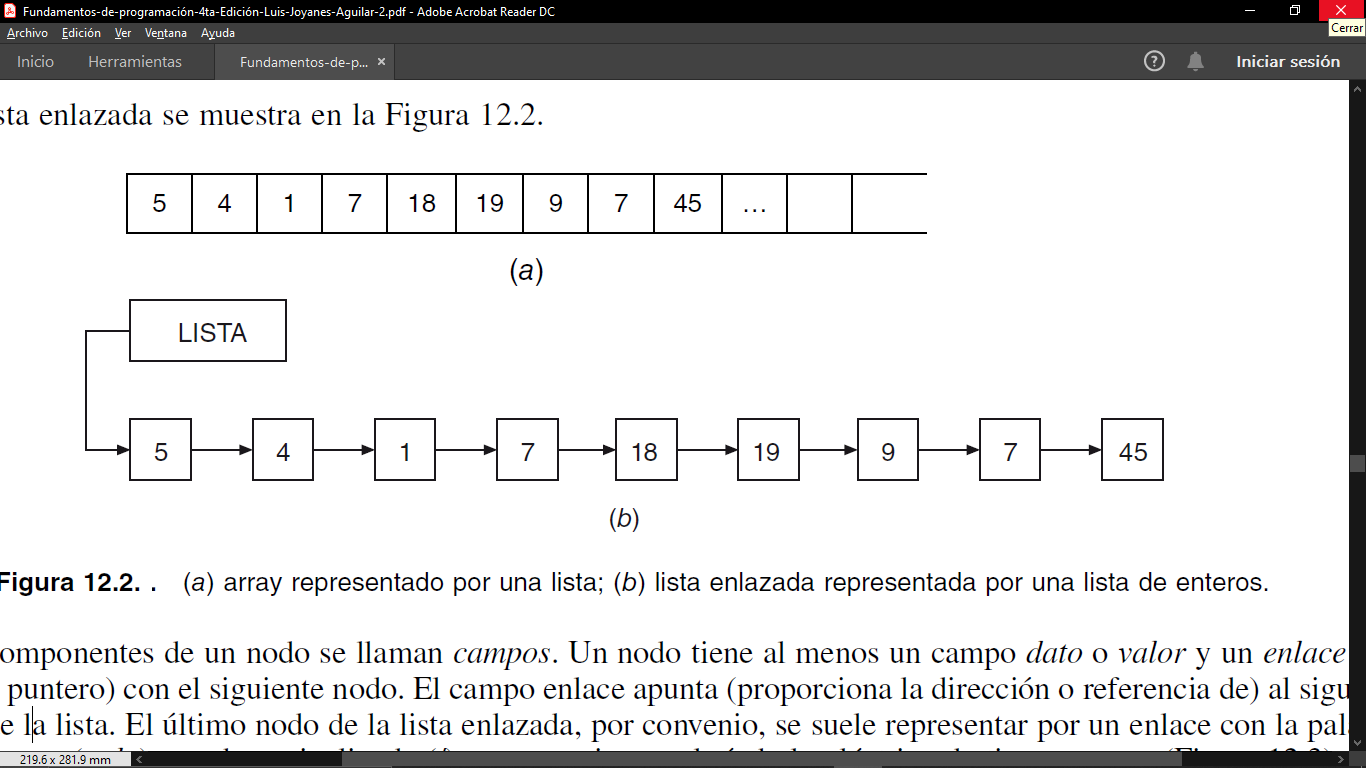
Una *lista enlazada* o *encadenada* es un conjunto de elementos en los que cada elemento contiene la posición —o dirección— del siguiente elemento de la lista. Cada elemento de la lista enlazada debe tener al menos dos campos: un campo que tiene el valor del elemento y un campo (enlace, *apuntador*) que contiene la posición del siguiente elemento, es decir, su conexión, enlace o encadenamiento. Los elementos de una lista son enlazados por medio de los campos enlaces.

Las listas enlazadas tienen una terminología propia que se suele utilizar normalmente. Primero, los valores se almacenan en un *nodo,* figura siguiente:



Una lista enlazada se muestra en la figura siguiente:





(*a*) array representado por una lista; (*b*) lista enlazada representada por una lista de enteros.

Los componentes de un nodo se llaman *campos*. Un nodo tiene al menos un campo *dato* o *valor* y un *enlace* (indicador, índice o puntero) con el siguiente nodo. El campo enlace apunta (proporciona la dirección o referencia de) al siguiente nodo de la lista. El último nodo de la lista enlazada, por convenio, se suele representar por un enlace con la palabra reservada NULL (*nulo*).

Un *puntero* (**apuntador**) es una variable cuyo valor es la dirección o posición de otra variable.

En las listas enlazadas no es necesario que los elementos de la lista sean almacenados en posiciones físicas adyacentes, ya que el puntero indica dónde se encuentra el siguiente elemento de la lista.

Una lista enlazada sin ningún elemento se llama *lista vacía*. Su puntero inicial o de cabecera tiene el valor nulo **(**NULL).

Una lista enlazada se define por:

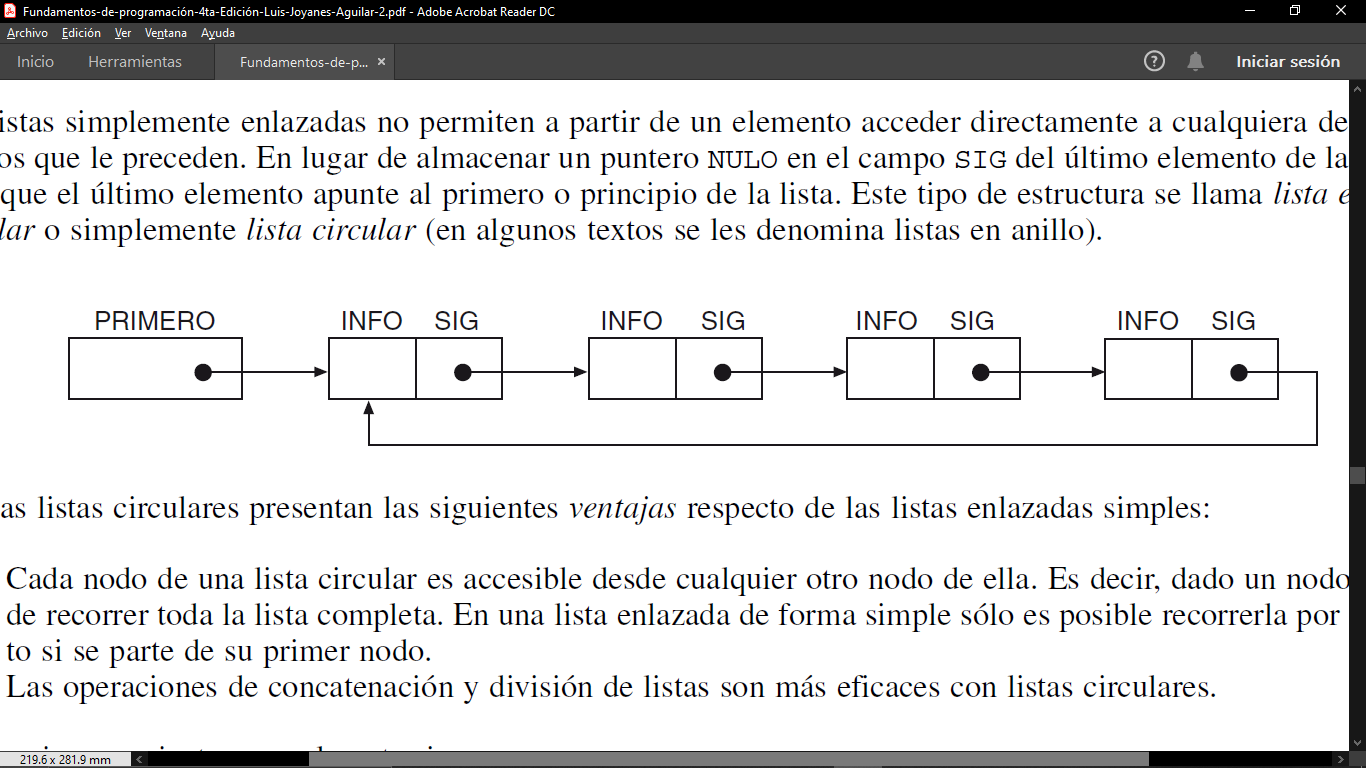
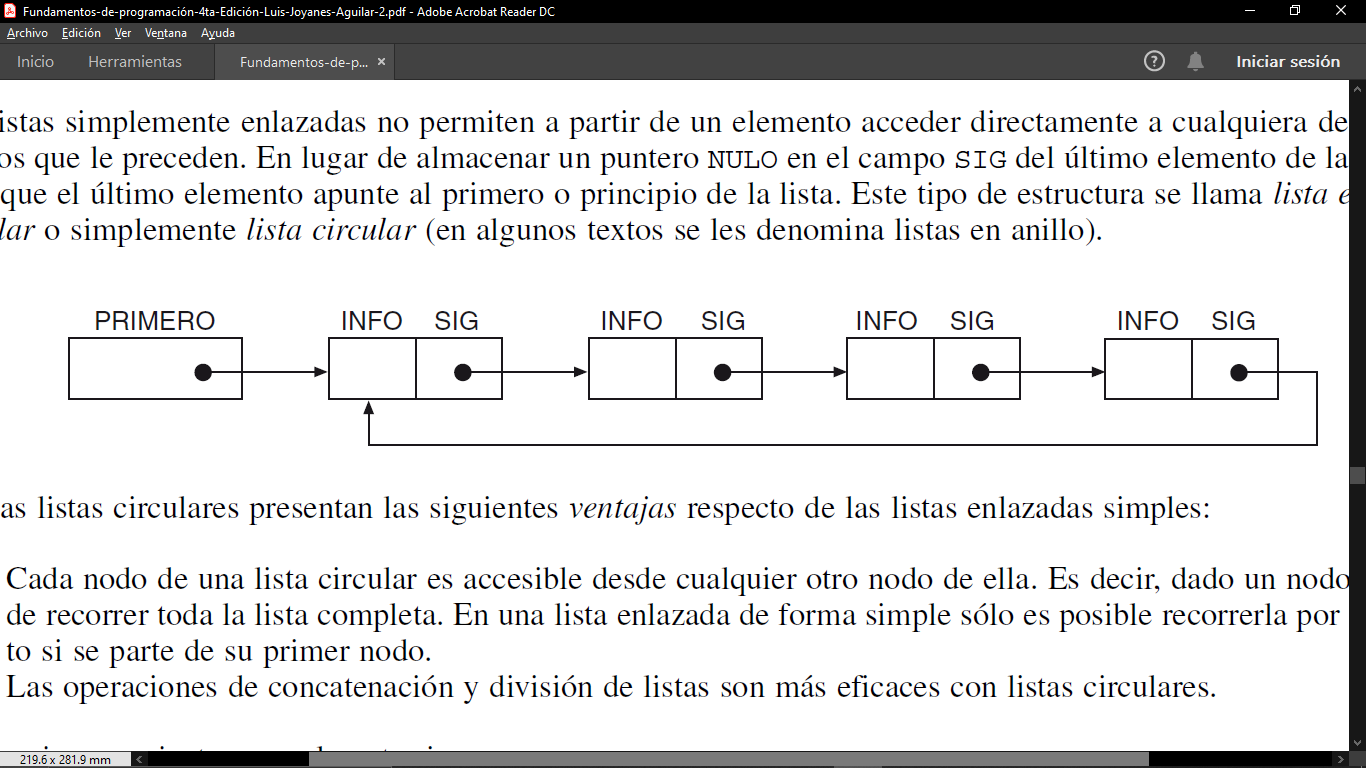
• El tipo de sus elementos: campo de información (datos) y campo enlace (*puntero o apuntador*).

• Un *puntero de cabecera* que permite acceder al primer elemento de la lista.

• Un medio para detectar el último elemento de la lista: *puntero nulo* (NULL).

**LISTAS CIRCULARES**

Las listas simplemente enlazadas no permiten a partir de un elemento acceder directamente a cualquiera de los elementos que le preceden. En lugar de almacenar un puntero NULO en el campo sig del último elemento de la lista, se hace que el último elemento apunte al primero o principio de la lista. Este tipo de estructura se llama *lista enlazada circular* o simplemente *lista circular o lista circular simplemente enlazada* (en algunos textos se les denomina listas en anillo).



Las listas circulares presentan las siguientes *ventajas* respecto de las listas enlazadas simples:

• Cada nodo de una lista circular es accesible desde cualquier otro nodo de ella. Es decir, dado un nodo se puede recorrer toda la lista completa. En una lista enlazada de forma simple sólo es posible recorrerla por completo si se parte de su primer nodo.

• Las operaciones de concatenación (unión) y división de listas son más eficaces con listas circulares.

Los *inconvenientes*, por el contrario, son:

• Se pueden producir lazos o ciclos infinitos. Una forma de evitar estos bucles infinitos es disponer de un nodo especial que se encuentre permanentemente asociado a la existencia de la lista circular. Este nodo se denomina *inicio principio o cabecera* de la lista.

El nodo inicio puede diferenciarse de los otros nodos en una de las dos formas siguientes:

• Puede tener un valor especial en su campo dato que no es válido como datos de otros elementos.

• Puede tener un índice o bandera *(flag)* que señale cuando es nodo inicio.

El campo de la información del nodo inicio no se utiliza.

**LISTAS DOBLEMENTE ENLAZADAS**

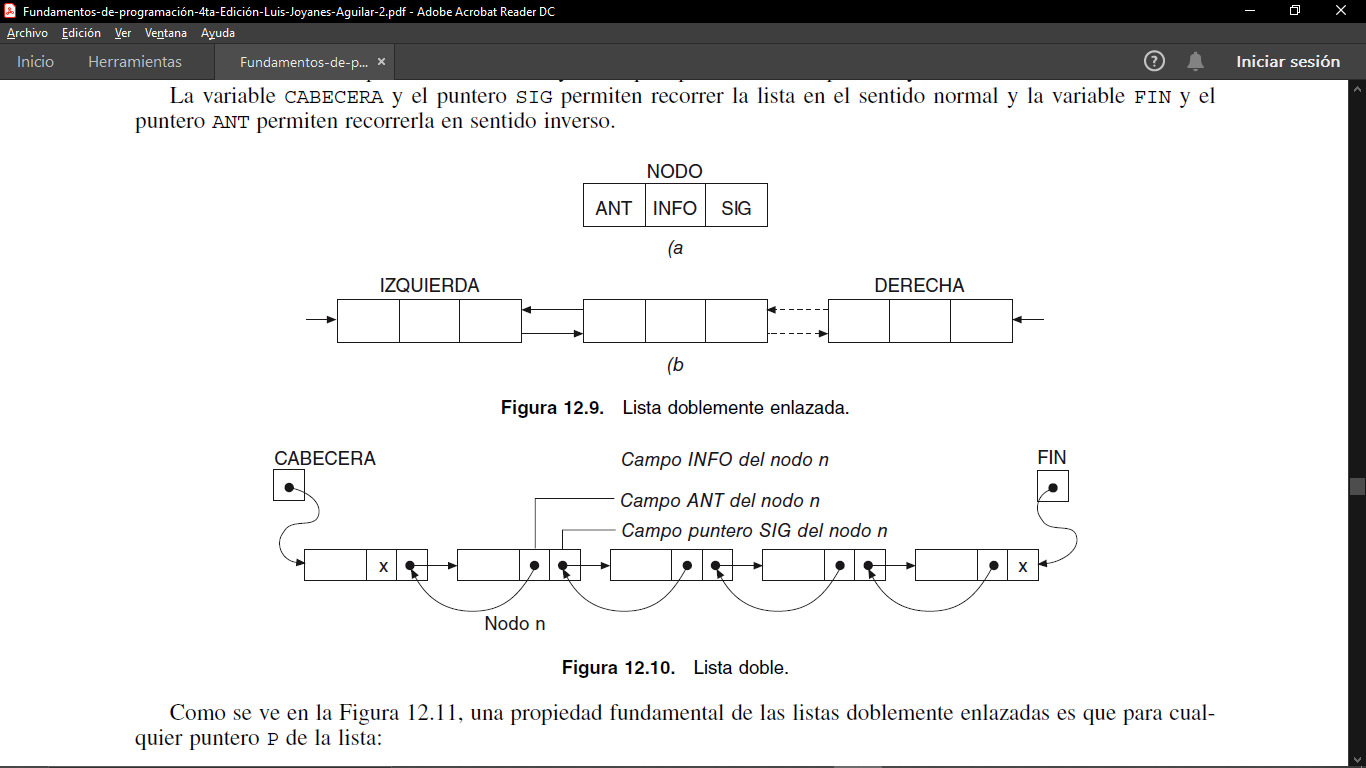
En las listas lineales estudiadas anteriormente el recorrido de ellas sólo podía hacerse en un único sentido: *de izquierda a derecha* (principio a final). En numerosas ocasiones se necesita recorrer las listas en ambas direcciones.

Las listas que pueden recorrerse en ambas direcciones se denominan *listas doblemente enlazadas*. En estas listas cada nodo consta del campo dato de información y dos campos de enlace o punteros: ANTERIOR(ANT) y SIGUIENTE(SIG) que apuntan hacia adelante y hacia atrás. Como cada elemento tiene dos punteros, una lista doblemente enlazada ocupa más espacio en

-memoria que una lista simplemente enlazada para una misma cantidad de información.

La lista necesita dos punteros inicio1 e inicio2 que apuntan hacia el primero y último nodo.

La variable inicio1 y el puntero SIG permiten recorrer la lista en el sentido normal y la variable inicio2 y el puntero ANT permiten recorrerla en sentido inverso.



**PILAS**

Una *pila (stack)* es un tipo especial de lista lineal en la que la inserción y borrado de nuevos elementos se realiza sólo por un extremo que se denomina *cima o tope (top)*.

La pila es una estructura con numerosas analogías en la vida real: una pila de platos, una pila de monedas, una pila de cajas de zapatos, una pila de camisas, una pila de bandejas, etc.

Dado que las operaciones de insertar y eliminar se realizan por un solo extremo (el superior), los elementos sólo pueden eliminarse en orden inverso al que se insertan en la pila. El último elemento que se pone en la pila es el primero que se puede sacar; por ello, a estas estructuras se les conoce por el nombre de **LIFO** (*Last-In, First-Out,* último en entrar, primero en salir).

Las operaciones más usuales asociadas a las pilas son:

"push": *Meter, poner o apilar:* operación de insertar un elemento en la pila.

"pop": *Sacar, quitar o desapilar:* operación de eliminar un elemento de la pila.

**Aplicaciones de las pilas**

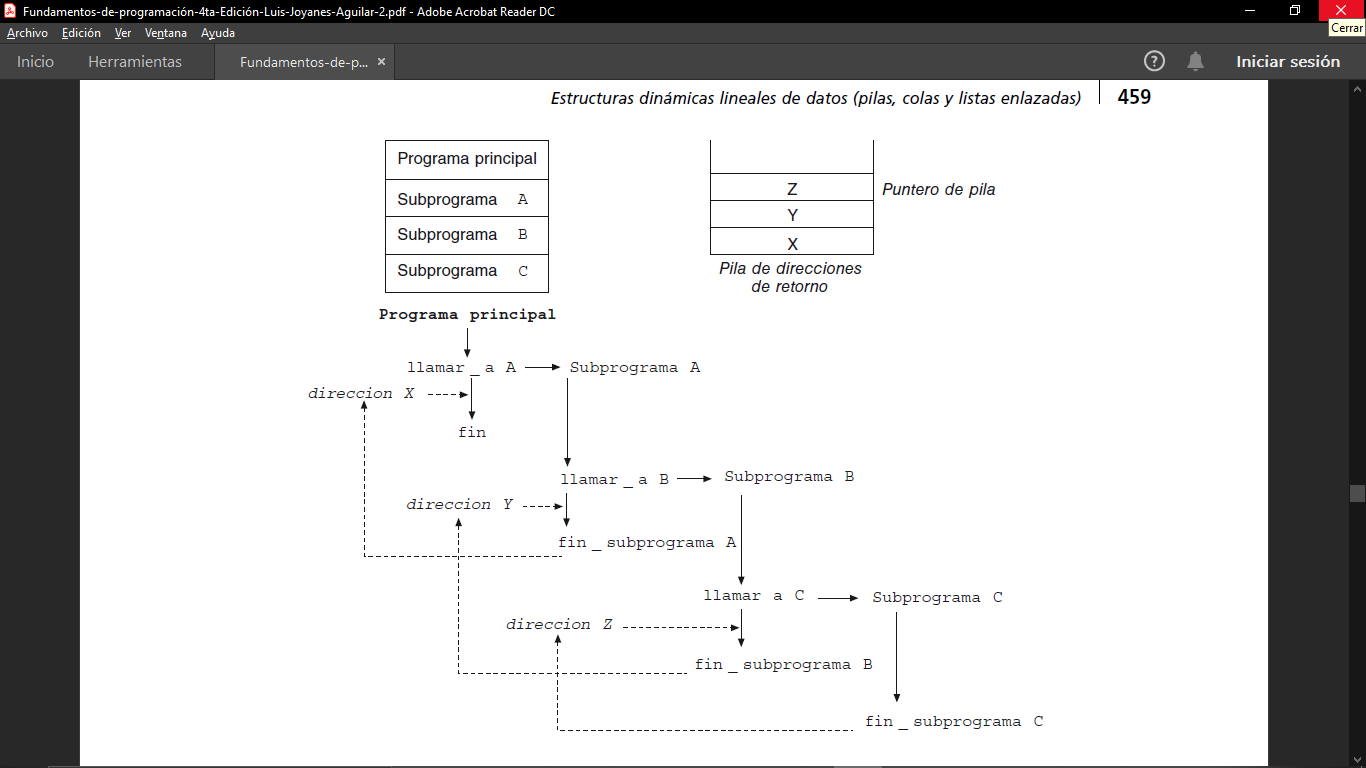
Las pilas son utilizadas ampliamente para solucionar una amplia variedad de problemas. Se utilizan en compiladores, sistemas operativos y en programas de aplicación. Veamos algunas de las aplicaciones más interesantes.

***Llamadas a subprogramas***

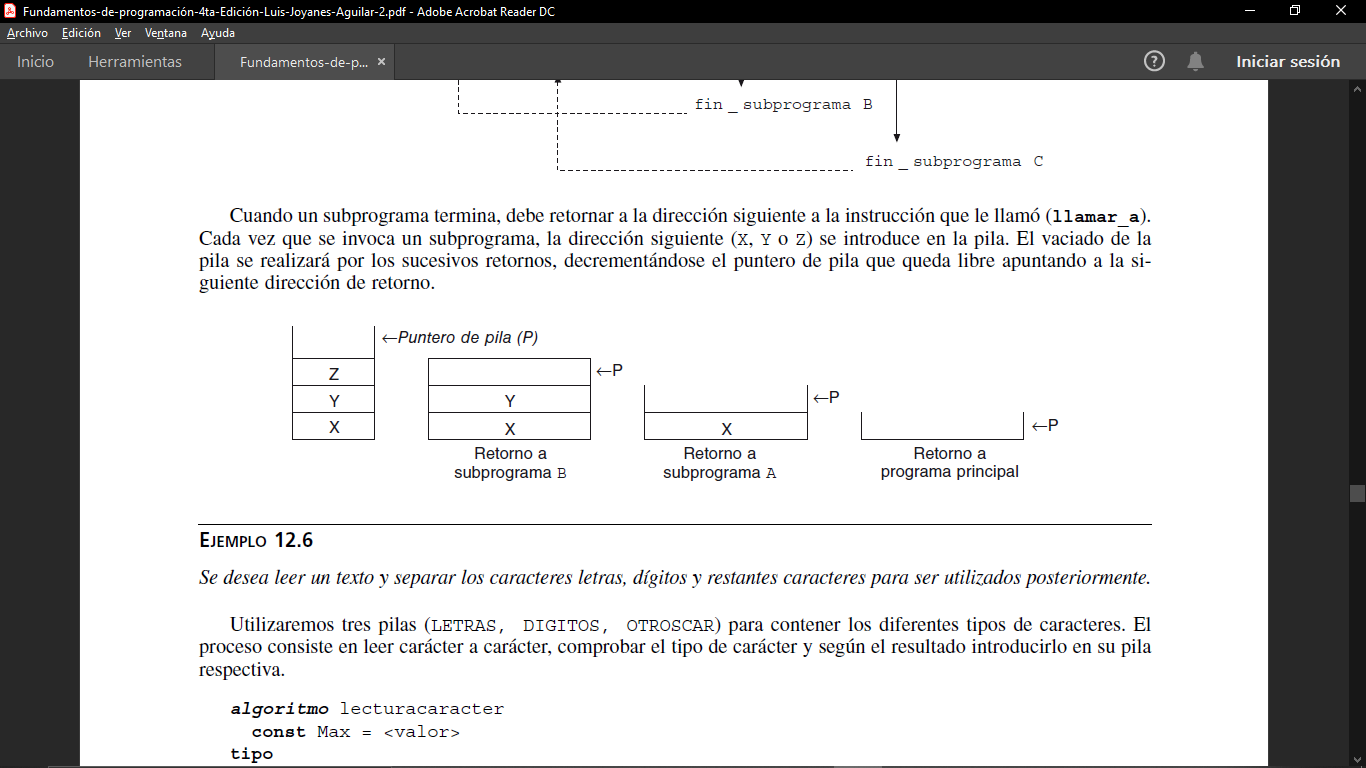
Cuando dentro de un programa se realizan llamadas a subprogramas, el programa principal debe recordar el lugar donde se hizo la llamada, de modo que pueda retornar allí cuando el subprograma se haya terminado de ejecutar.

Supongamos que tenemos tres subprogramas llamados A, B y C, y supongamos también que A invoca a B y B invoca a C. Entonces B no terminará su trabajo hasta que C haya terminado y devuelto su control a B. De modo similar, A es el primero que arranca su ejecución, pero es el último que la termina, tras la terminación y retorno de B.

Esta operación se consigue disponiendo las direcciones de retorno en una pila.



Cuando un subprograma termina, debe retornar a la dirección siguiente a la instrucción que le llamó (**llamar\_a**). Cada vez que se llama un subprograma, la dirección siguiente (X, Y o Z) se introduce en la pila. El vaciado de la pila se realizará por los sucesivos retornos, decrementándose el puntero de pila que queda libre apuntando a la siguiente dirección de retorno.



**COLAS**

Las colas son otro tipo de estructura lineal de datos similar a las pilas, diferenciándose de ellas en el modo de insertar/ eliminar elementos.

Una *cola (queue)* es una estructura lineal de datos en la que las *eliminaciones* se realizan al principio de la lista, *frente (front)*, y las *inserciones* se realizan en el otro extremo, *final (rear)*. En las colas el elemento que entró el primero sale también el primero; por ello se conoce como listas **FIFO** (*First-In, First-Out*, “primero en entrar, primero en salir”). Así, pues, la diferencia con las pilas reside en el modo de entrada/salida de datos; en las colas las inserciones se realizan al final de la lista, no al principio. Por ello las colas se usan para almacenar datos que necesitan ser procesados según el orden de llegada.

En la vida real se tienen ejemplos numerosos de colas: la cola de un autobús, la cola de un cine, una caravana de coches en una calle, etc. En todas ellas el primer elemento (pasajero, coche, etc.) que llega es el primero que sale.

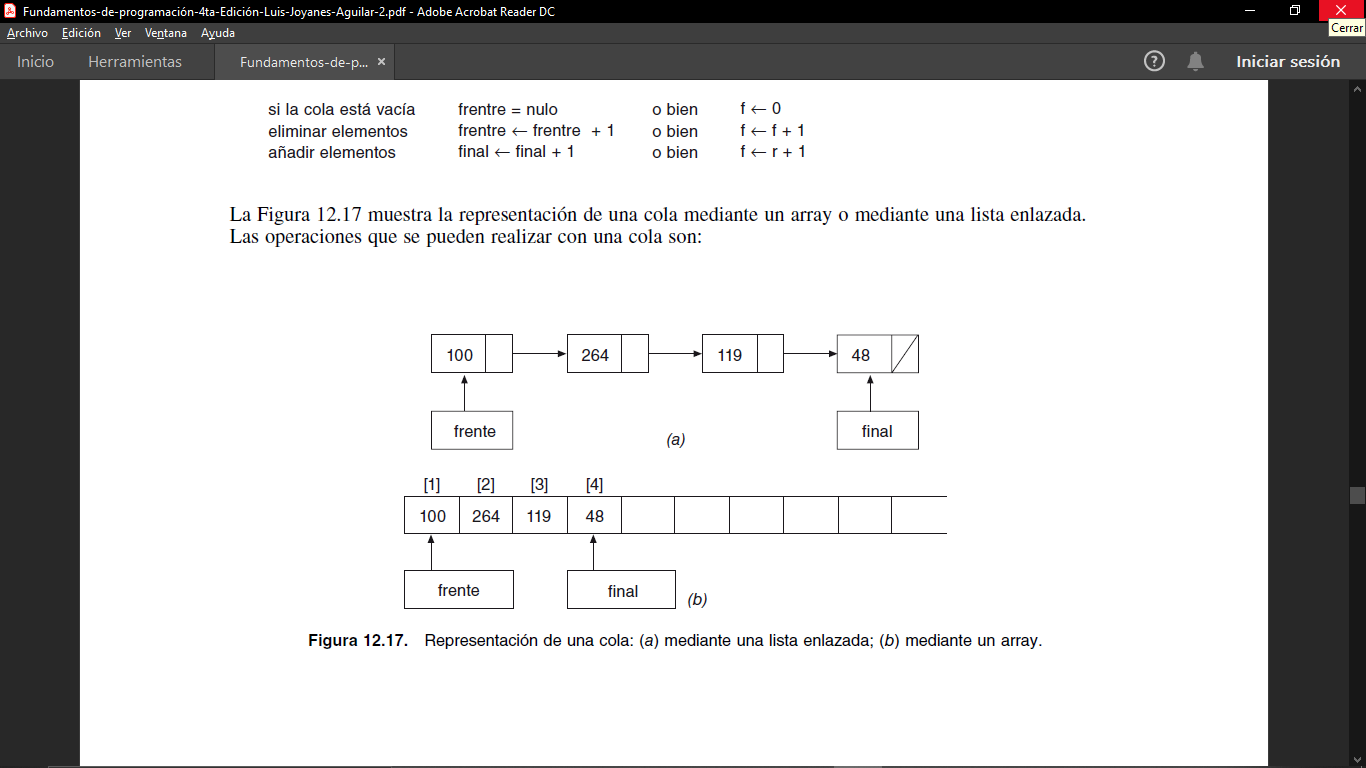
En informática existen también numerosas aplicaciones de las colas. Por ejemplo, en un sistema de tiempo compartido suele haber un procesador central y una serie de periféricos compartidos: discos, impresoras, etc. Los recursos se comparten por los diferentes usuarios y se utiliza una cola para almacenar los programas o peticiones de los diferentes usuarios que esperan su turno de ejecución. El procesador central atiende —normalmente— por riguroso orden de llamada del usuario; por tanto, todas las llamadas se almacenan en una cola. Existe otra aplicación muy utilizada que se denomina *cola de prioridades*; en ella el procesador central no atiende por riguroso orden de llamada, aquí el procesador atiende por prioridades asignadas por el sistema o bien por el usuario, y sólo dentro de las peticiones de igual prioridad se producirá una cola.

**Representación de las colas**

Las colas se pueden representar por listas enlazadas o por arreglos.

Se necesitan dos apuntadores: *frente(f)* y *final(r)*, y la lista o arreglo de *n* elementos (LONGMAX).

La figura muestra la representación de una cola mediante un arreglo o mediante una lista enlazada.



Representación de una cola: (*a*) mediante una lista enlazada; (*b*) mediante un arreglo.

Las operaciones que se pueden realizar con una cola son:

• Acceder al primer elemento de la cola.

• Añadir un elemento al final de cola.

• Eliminar el primer elemento de la cola.

• Vaciar la cola.

• Verificar el estado de la cola: vacía o llena.

**RESUMEN**

Una **lista lineal** es una lista en la que cada elemento tiene un único sucesor. Las operaciones típicas en una lista lineal son: inserción, supresión, recuperación y recorrido.

Una **lista enlazada** es una colección ordenada de datos en los que cada elemento contiene la posición (dirección) del siguiente elemento. Es decir, cada elemento (nodo) de la lista contiene dos partes: datos y enlace (apuntador).

Una **lista simplemente enlazada** contiene sólo un enlace a un sucesor único a menos que sea el último, en cuyo caso no se enlaza con ningún otro nodo. Cuando se desea insertar un elemento en una lista enlazada, se deben considerar dos casos: añadir al principio y añadir en el interior o añadir al final. Si se desea eliminar un nodo de una lista se deben considerar dos casos: eliminar el primer nodo y eliminar cualquier otro nodo. El recorrido de una lista enlazada implica visitar cada nodo de la lista y procesar en su caso.

Una **lista doblemente enlazada** es una lista en la que cada nodo tiene un puntero a su sucesor y otro a su predecesor.

Una **lista enlazada circularmente** es una lista en la que el enlace del último nodo apunta al primero de la

lista.

Una **pila** es una estructura de datos tipo **LIFO** (***last-in, first-out,*** último en entrar, primero en salir) en la que los datos se insertan y eliminan por el mismo extremo que se denomina *cima de la pila.* Se definen diferentes operaciones: crear, apilar, desapilar, pilaVacía, pila- Llena, cimaPila.

Una **cola** (**FIFO,** first-in, first-out) es una lista lineal en la que los datos se pueden insertar por un extremo denominado *Inicio* y se elimina o borra por el otro extremo denominado *Cola* o *Final.* Las operaciones básicas de una cola son: poner, quitar, frenteCola y Colavacia, Colallena.

Las pilas y las colas se pueden implementar mediante arreglos y mediante listas enlazadas.