	Carátula para entrega de prácticas	
Facultad de Ingeniería		Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Edgar Tista Garcia

Asignatura: Estructuras de Datos y Algoritmos I

Grupo: 2

No. de práctica(s): 7

Integrante(s): Álvarez López Carlos Manuel

No. de lista o brigada: 2

Semestre: 2023-2

Fecha de entrega: Sábado 21 de Abril

Observaciones:

CALIFICACIÓN: _____

Objetivo: Revisarás las definiciones, características, procedimientos y ejemplos de las estructuras lineales Lista simple y Lista circular, con la finalidad de que comprendas sus estructuras y puedas implementarlas.
Objetivo de clase: Analizar y comprender implementaciones de lista ligada y lista ligada circular así como la encapsulación de sus respectivas funciones. El alumno será capaz de agregar nuevas funcionalidades a dichas bibliotecas e implementar listas de tipo de dato abstracto.

Actividad 1: Revisión y análisis de las bibliotecas proporcionadas
Se realiza un análisis de las funciones de las bibliotecas proporcionadas, tanto de lista circular como de lista simple.

Al ambas ser listas ligadas, la mayoria de sus funciones son similares, con la particularidad de que con la lista circular, en las funciones referentes al enlistado se tiene que tomar en cuenta

- Lista crearLista();

Devuelve una variable de tipo lista, y no recibe parámetros.

- Simple

Se crea una estructura de tipo lista y se iguala la propiedad "head" a NULL, para después devolver la estructura.

- Circular

Se crea una estructura de tipo lista, a la cual sus propiedades "head" y tamaño se igualan a NULL y 0 respectivamente. Finalmente se devuelve la estructura lista.

- void print_list(Lista);

Con esta función no se retorna nada, y solo se recibe como parámetro una estructura de lista.

- Simple

Se comprueba si la lista está vacía verificando si la propiedad "head" es igual a NULL. En caso contrario se declara un apuntador de tipo nodo y que se asocia a la propiedad "head" ; después ese apuntador se ocupa para imprimir y recorrer cada elemento de la lista, igualandolo a su elemento siguiente después de imprimir en pantalla, mientras que este apuntador sea distinto de 0.

- Circular

Se comprueba si la lista está vacía verificando si la propiedad "head" es igual a NULL. En caso contrario se declara un apuntador de tipo nodo y que se asocia a la propiedad "head" , además de una variable tamaño que indica el número de elementos. Ocupando un ciclo while, con la condición mientras que tamaño sea mayor de 0; se imprime la información del elemento, y el apuntador se iguala a su elemento siguiente.

- void addPrincipioLista(Lista*,int);

Esta función es de tipo void y como parámetros recibe un apuntador a lista y una variable de tipo entero.

- Simple

Se crea un apuntador de tipo nodo, al cual se le aparta espacio en la memoria ocupando malloc, para después igualar su valor al parámetro de tipo entero que recibe la función, y su propiedad next a la propiedad "head" de la lista, y finalmente la igualar la propiedad de la cola "head" al nuevo nodo.

- Circular

Se crean dos apuntadores de tipo nodo, node(para el nodo a añadir) y temp(un auxiliar para el manejo de la propiedad "head"); Al igual que en la lista simple se crea el nodo con malloc y se le define su propiedad valor. Lo siguiente es comprobar si la lista está vacía, para lo que la propiedad next del nodo se iguala a "head" ; en caso de que la lista no esté vacía, el next del nodo se iguala a el "head" de la lista, y temp se iguala a "head" ; para que después temp avance igualandose a su siguiente elemento hasta que su propiedad next sea igual a la propiedad "head" (Sea el último elemento), y se iguale su propiedad next al node. Finalmente ya sea que la lista esté vacía o no, la propiedad "head" se iguala a node y se realiza un incremento a la propiedad tamaño de la lista.

- void addFinalLista(Lista*,int);

Función de tipo void que recibe como parámetros un apuntador a lista y una variable de tipo entero.

- Simple

Si la lista está vacía:

> Se crea un nodo con memoria dinámica y se le define su propiedad valor con el parámetro de tipo entero que recibe la función. Además de igualar su propiedad next a NULL, y el "head" de la lista al nuevo nodo.

Si la lista no está vacía:

> Se crea un nodo auxiliar el cual avanza con un ciclo while hasta que su propiedad siguiente sea igual a 0(Llegue al final de la lista). Posteriormente se crea el nodo a añadir como en la función de lista simple, adicionando que la propiedad next del nodo auxiliar se iguala al nuevo nodo.

- Circular

Se crea una variable de tipo entero, posición, para guardar el número de elementos de la lista.

Si la lista está vacía:

> Se crea el nodo a añadir con memoria dinámica y se le definen sus valores como en la función de lista simple.

Si la lista no está vacía:

Se crea un nodo auxiliar para recorrer la lista, el cual avanza en un ciclo while, mientras que la variable posición sea distinta de 1, dentro del ciclo while también se realiza un decremento de la variable posición.

Se crea el nodo a añadir con memoria dinámica como en la función de lista simple y se iguala su propiedad next a "head" .

Finalmente, independientemente de si la cola está vacía, se realiza un incremento a la variable tamaño de la lista.

- void borrarPrimero(Lista*);

Es una función de tipo void que recibe como parámetro un apuntador a lista.

- Simple

Se comprueba si la lista está vacía, para imprimir un mensaje indicandolo; en caso contrario, se crean dos apuntadores a nodo, “nuevo_head” y “tmp”. ”tmp” se iguala al “head” de la lista, para después igualar “nuevo_head” al siguiente elemento de “tmp”. Enseguida se libera el “head” de la lista y se vuelve a asignar el “head” de la misma a “nuevo_head”.

- Circular

Esta función para una lista circular es igual que para una lista simple, solo que contiene la inclusión de una línea más de código para cuando la lista si tiene elementos, la cual realiza un decremento de la propiedad “tamano” de la lista.

- void borrarUltimo(Lista*);

Esta función no retorna nada, y solo recibe un apuntador a lista como parámetro.

- Simple

Primero se crea un nodo “temp” asociado a la propiedad “head” de la lista. Después se procede a verificar si la lista tiene elementos, que en caso de que no se imprime en pantalla un mensaje indicandolo; en caso de que la lista tenga elementos, se entra en un if que verifica si la lista solo tiene un elemento al evaluar si la propiedad “next” de “temp” es igual a NULL, para lo cual se libera “head” de la lista(el único elemento) y se iguala a NULL. En caso de que la lista tenga más de un nodo, se crea otro nodo temporal llamado “current” que se iguala al “head” de la lista, “current” avanza por la lista en un ciclo while hasta que el “next” del “next” de “current” sea distinto de NULL, o en otras palabras qué “current” esté en la penúltima posición de la lista. Finalmente se libera el elemento siguiente de “current” y su propiedad “next” se iguala a NULL.

- Circular

Esta función tiene las mismas instrucciones que la de la lista simple, hasta el caso en que la lista tiene más de un nodo, para lo cual se crean dos apuntadores a nodo, “current” y “current2”, los cuales se igualan al “head” de la lista y al “next” de “current” respectivamente. En un ciclo while, ambos apuntadores van avanzado de elemento en elemento en la lista, hasta que el “next” de “current2” es distinto de NULL, lo cual indica que “current2” está apuntando al último elemento de la lista. Finalmente se libera “current2” y se iguala “current->next” a NULL.

- int primerElemento(Lista);

Retorna un entero y solo recibe una lista en paso por valor.

- void eliminarLista(Lista*)

Esta función que recibe como único parámetro un apuntador a lista, llama a la función “borrarPrimero”, hasta que la propiedad head de la lista sea igual a NULL, vaciando la lista y liberando cada nodo.

- Simple

Únicamente retorna el valor guardado en la propiedad “val” del “head” de la lista.

En algunas funciones de eliminación de elementos, se modificaron los archivos recibidos, para que primero se libere del nodo a eliminar y después se iguale el apuntador asociado a dicho nodo a NULL.

Actividad 2: Implementación lista ligada simple

a) Función int buscar(Lista, int)

Para esta función que retorna un entero indicando la posición del nodo que contiene el valor buscado, y que recibe una lista, se necesito de un apuntador de tipo nodo llamado “tmp” y una variable de tipo entero para guardar la posición en la que se encuentra el elemento buscado, aunado a un ciclo “do-while” para recorrer la lista.

Una vez que se declara el apuntador al nodo y se iguala a la propiedad head de la lista, y la variable de tipo entero, se evalúa si la lista está vacía al comprobar que la propiedad head de la lista sea distinta de NULL, en caso de que la lista esté vacía se imprime un mensaje indicandolo y se retorna -1. En el caso de que la lista tenga elementos, se entra en un ciclo do-while que terminará hasta que la propiedad next de tmp sea igual a NULL (Se haya recorrido toda la lista), en el cual primero se comprobará que el valor de tmp sea igual que el valor buscado, que en caso de ser verdadero la función retorna la variable que guarda la posición en la que se

encuentra tmp; en caso de no se encuentre el valor buscado tmp se iguala a su siguiente y la variable de posición se incrementa. Este proceso se repite hasta acabar la lista, y en caso de que no se encuentre el valor buscado en ningún nodo de la lista, entonces la función retorna -1.

b) void addN-esimo(Lista*,int,int)

Esta función recibe un apuntador a lista y dos variables de tipo entero, una para el valor del nodo a añadir, y otro para la posición en la que se introduce el nodo. Adicionalmente se hace uso de dos apuntadores a nodo, “tmp” y “nuevo”, y la función malloc para apartar dinámicamente la memoria necesaria para el nuevo nodo.

La función en caso de tener elegir la posición 0(inicio de la lista), llama a la función “addPrincipioLista”. Si la posición recibida es distinta de 0, entonces se avanza el apuntador a nodo “tmp” tantas las posiciones recibidas menos 1, igualando “tmp” a su propiedad “next”, para que se posicione un nodo antes de la posición indicada; si al avanzar por la función, la propiedad next de tmp es igual a NULL, entonces se finaliza la función y se imprime un mensaje indicando que la posición introducida está fuera del rango de la lista.

Ya teniendo posicionado tmp, se declara el apuntador al nodo para el nuevo nodo, y se aparta dinámicamente la memoria para el mismo, asignando el valor recibido como parámetro y su propiedad next a la propiedad next de tmp, y finalmente igualando la propiedad next de tmp al nuevo nodo.

- void borrarN-esimo(Lista*,int)

Esta función recibe un apuntador a lista y la posición de la que se borrara el nodo. En la función se ocupan dos apuntadores a nodo, “tmp” y “kill”.

Esta función es bastante similar a “addN-ésimo”, ya que si la posición recibida como parámetro es igual a 0, la función llama a la función “borrarPrimero”, y en caso contrario, continúa avanzando “tmp” hasta una posición antes de la posición a eliminar, además de que en caso de que la posición indicada no esté en la lista se termina la función y se imprime un mensaje indicandolo. La diferencia entre ambas funciones radica que una vez que el apuntador “tmp” está asociado al elemento en la posición anterior a la posición a eliminar, si su propiedad next es igual a NULL se llama a la función “borrarUltimo”, y en caso de que sea distinto de NULL, el apuntador kill se iguala al nodo siguiente de “tmp”(elemento en la posición a eliminar), la propiedad next de tmp se iguala al next.next(elemento siguiente al elemento a eliminar), y finalmente se libera kill.

c) void eliminarMayores(Lista*, int)

La función recibe como parámetros un apuntador a lista, además de un entero el cual será el mayor. Adicionalmente ocupará un apuntador a nodo, “tmp”.

Primero se crean dos apuntadores a nodo, “tmp” que se iguala al head de la lista, y aux, que se ocupará más adelante para avanzar los apuntadores cuando se elimine algún nodo; además de las variables de tipo entero eliminados y pos, las cuales ambas se igualaran a 0. Después dentro de un ciclo do-while se iguala aux a tmp, y tmp se iguala a su propiedad next, después se evalúa si el valor de aux es mayor al parámetro recibido como mayor, en cuyo caso se llama a la función borrarN-esimo para que borre de la lista ese nodo; la posición se determina restando la variable eliminados(número de elementos eliminados) a la variable pos(posición del elemento a eliminar respecto a la lista original), para después realizar un incremento a la variable eliminados. Después independientemente de si se haya eliminado algún nodo, se incrementa la variable posición. Estas instrucciones se repiten hasta que la propiedad next de tmp sea igual a NULL, indicando que se ha llegado al final de la lista.

Finalmente se imprime en pantalla un mensaje indicando cuántos elementos fueron eliminados de la lista, o si no se eliminó ninguno.

d)

Para este inciso se creó una lista ocupando la función de los archivos dados, además de 3 variables de tipo entero para guardar la opción elegida, la posición, y el valor de los nodos.

Para el usuario se imprimen en pantalla las posibles acciones a realizar con la lista, y se registra la opción elegida, para después ejecutar las acciones pertinentes mediante un switch; solicitando valor del nodo a crear para cuando se añaden nodos, posición en el caso de tratar con n-esimos, o valor cuando se realice una búsqueda o cuando se borren mayores. En el caso de la búsqueda se implementó que cuando la función retorna un -1, se imprima en pantalla que no se pudo encontrar el valor buscado.

Esto se repite mientras que el usuario introduzca una de las opciones mostradas, en caso contrario se llama a la función “eliminarLista” y se finaliza la ejecución.

Agregar

```
Los elementos de la lista son:
1
2
3
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
1
Introduce valor: 0
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
2
Introduce valor: 5
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
3
Introduce valor: 4
Indica posicion: 4
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
9
Los elementos de la lista son:
0
1
2
3
4
5
```

Eliminar

```
Los elementos de la lista son:
0
1
2
3
4
5
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
6
Indica posicion: 4
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
4
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
5
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
9
Los elementos de la lista son:
1
2
3
```

Buscar

```
Los elementos de la lista son:
8
1
2
3
2
8
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
7
Introduzca valor a buscar: 2
Se encontro en la posicion: 2
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
7
Introduzca valor a buscar: 9
No se encontro el valor en la lista
```

Eliminar mayores

```
Los elementos de la lista son:
8
1
3
2
8
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
8
Indica el mayor: 2
Se eliminaron 3 elementos
Eliga la accion a realizar:
1.Agregar al principio  2.Agregar al final
3.Agregar al n-├®simo   4.Eliminar al principio
5.Eliminar al final     6.Eliminar al n-├®simo
7.Buscar por ├¡ndice    8.Eliminar mayores
9.Imprimir lista
9
Los elementos de la lista son:
1
2
Eliga la accion a realizar:
```

Actividad 3: Lista ligada circular

Principales cambios:

- Manejo de la propiedad “tamano” de la lista circular

  En el caso de añadir al final de la función se realiza un incremento de la propiedad, y para borrar se realiza un decremento

- Caso en el que la posición introducida excede el tamaño de la lista

Con la lista simple cuando la posición recibida excede el tamaño de la lista la función no realiza cambios en la lista, pero en el caso de la lista circular, ya que una vez que se llega al final, se puede continuar volviendo al inicio, se optó que la función siguiera adelantado el apuntador “tmp” a lo largo de la lista hasta haber avanzado las posiciones especificadas.

- void addN_esimo(Lista*,int,int)

Al igual que con la función de lista simple, esta función recibe un apuntador a lista y dos variables de tipo entero, una para el valor del nodo a añadir, y otro para la posición en la que se introduce el nodo. Adicionalmente se hace uso de dos apuntadores a nodo, “tmp” y “nuevo”, y la función malloc para apartar dinámicamente la memoria necesaria para el nuevo nodo.

La función en caso de tener elegir la posición 0 (inicio de la lista) o que la propiedad head de la lista sea igual a NULL(la lista esté vacía), llama a la función “addPrincipioLista”. Si la posición recibida es distinta de 0, entonces se avanza el apuntador a nodo “tmp” tantas las posiciones recibidas menos 1, igualando “tmp” a su propiedad “next”, para que se posicione un nodo antes de la posición indicada. Dado que se está tratando con una lista circular, no hace falta especificar un caso en el que la propiedad next de tmp sea igual a NULL, ya que si la posición dada excede el tamaño de la lista, tmp podrá continuar hasta haber avanzado las posiciones especificadas.

Ya teniendo posicionado tmp, se declara el apuntador al nodo para el nuevo nodo, y se aparta dinámicamente la memoria para el mismo, asignando el valor recibido como parámetro y su propiedad next a la propiedad next de tmp, y finalmente igualando la propiedad next de tmp al nuevo nodo, e incrementando la propiedad de tamaño de la lista.

- void borrarN_esimo(Lista*, int)

Esta función recibe un apuntador a lista y la posición de la que se borrara el nodo. En la función se ocupan dos apuntadores a nodo, “tmp” y “kill”.

Esta función es bastante similar a “addN-ésimo”, ya que si la posición recibida como parámetro es igual a 0, la función llama a la función “borrarPrimero”, y en caso contrario, continúa avanzando “tmp” hasta una posición antes de la posición a eliminar, en caso de que la posición indicada exceda el tamaño de la lista, tmp seguirá avanzando hasta que se haber avanzado las posiciones indicadas. La diferencia entre ambas funciones radica que una vez que el apuntador “tmp” está asociado al elemento en la posición anterior a la posición a eliminar, y el apuntador kill se iguala al nodo siguiente de “tmp”(elemento en la posición a eliminar), la propiedad next de tmp se iguala al next.next(elemento siguiente al elemento a eliminar), finalmente se libera kill, y se realiza un decremento a la propiedad tamaño de la lista.

Añadir

```
Los elementos de la lista son:
1
2
3
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
1
Introduce valor: 0
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
2
Introduce valor: 5
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
3
Introduce valor: 4
Introduce poscion: 4
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
7
Los elementos de la lista son:
0
1
2
3
4
5
```

Borrar

```
Los elementos de la lista son:
0
1
2
3
4
5
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
6
Indica la poscion: 4
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
5
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
4
Indique la accion a realizar:
1.Añadir al principio
2.Añadir al final
3.Añadir al n-esimo
4.Eliminar al principio
5.Eliminar al final
6.Eliminar al n-esimo
7.Imprimir lista
7
Los elementos de la lista son:
1
2
3
```

Actividad 4: Lista de un tipo de dato abstracto

Para esta actividad se tomó el archivo de listaCirc.c y listaCirc.h y se modificaron con las especificaciones de la actividad. Para empezar se añadió la estructura “Computadora” con las propiedades especificadas en el documento de la práctica, además de funciones para crear e imprimir computadoras. Con respecto al nodo, a este se le modificó su propiedad “valor” para que en lugar de que albergara un entero, está albergará un apuntador a una estructura computadora; debido a esto todas las funciones de la lista se modificaron para que en el caso de añadir elementos, la función recibirá como parámetro un apuntador a computadora(para esto se hizo que la función “crearComputadora” devolviera un apuntador a computadora) y en el caso de eliminar nodo de la lista, se añadió una instrucción que libera la memoria asignada a la computadora, antes de liberar la memoria asociada al nodo.

Como se solicita en la actividad también se creó una función que sirve para que el usuario pueda recorrer la lista e indagar en las especificaciones de cada computadora si así lo desea. El realizar esta función no fue complicado, ya que solo se necesito un apuntador a nodo “tmp” como ya se ha ocupado durante la práctica, y una estructura switch para que el usuario pueda elegir sí desea mostrar la información de la computadora(para lo cual se llama a la función “imprimirComputadora”), avanzar al siguiente elemento de la lista(para lo cual se iguala tmp a su propiedad next), o salir del ciclo.

Añadir elemento y buscar por marca

```
Seleccione:
1.Agregar al inicio     2.Agregar al final
3.Agregar al n-esimo    4.Borrar primero
5.Borrar ultimo 6.Borrar n-esimo
7.Buscar por marca      8.Recorrer lista        2
Marca: Asus
Modelo: aa11
Ram[Gb]: 8
Procesador: Intel core
Almacenamiento[Gb]: 1024
Seleccione:
1.Agregar al inicio     2.Agregar al final
3.Agregar al n-esimo    4.Borrar primero
5.Borrar ultimo 6.Borrar n-esimo
7.Buscar por marca      8.Recorrer lista        1
Marca: HP
Modelo: bb-22
Ram[Gb]: 16
Procesador: 2048
Almacenamiento[Gb]: 2048
Seleccione:
1.Agregar al inicio     2.Agregar al final
3.Agregar al n-esimo    4.Borrar primero
5.Borrar ultimo 6.Borrar n-esimo
7.Buscar por marca      8.Recorrer lista        2
Marca: Lenovo
Modelo: cc-33
Ram[Gb]: 64
Procesador: 1024
Almacenamiento[Gb]: 1024
Seleccione:
1.Agregar al inicio     2.Agregar al final
3.Agregar al n-esimo    4.Borrar primero
5.Borrar ultimo 6.Borrar n-esimo
7.Buscar por marca      8.Recorrer lista        7
Introduzca marca a buscar: Lenovo
Se encontro en el elemento 2
```

Recorrer lista

```
7.Buscar por marca      8.Recorrer lista        8
Elemento 0
Seleccione:
0.Siguiente     1.Mostrar informacion
2.Salir
1
Marca: HP
Modelo: bb-22
Ram[Gb]: 16
Procesador: 2048
Almacenamiento[Gb]: 2048
Elemento 0
Seleccione:
0.Siguiente     1.Mostrar informacion
2.Salir
0
Elemento 1
Seleccione:
0.Siguiente     1.Mostrar informacion
2.Salir
1
Marca: Asus
Modelo: aa11
Ram[Gb]: 8
Procesador: Intel core
Almacenamiento[Gb]: 1024
Elemento 1
Seleccione:
0.Siguiente     1.Mostrar informacion
2.Salir
0
Elemento 2
Seleccione:
0.Siguiente     1.Mostrar informacion
2.Salir
1
Marca: Lenovo
Modelo: cc-33
Ram[Gb]: 64
Procesador: 1024
Almacenamiento[Gb]: 1024
```

Conclusión

Habiendo concluido con todos los ejercicios de la práctica, puedo afirmar que los objetivos de la misma fueron concretados, esta gracias a que se realizó un análisis de cada una de las funciones de ambas estructuras de datos, comprendiendo las propiedades e implicaciones de la implementación de cada una de ellas, aunado a la adición de otras funciones propuestas, concediendo la oportunidad de tratar con las propiedades y el manejo de los nodos de las listas.

Hablando de las actividades, además de que la primera es ideal para familiarizarse con las estructuras y comprender su funcionamiento, tanto la segunda como tercer actividad, son las que dotan de profundidad a la práctica y permiten que uno como alumno pueda manejar las estructuras, y lo obliguen a pensar en distintas soluciones para realizar las acciones solicitadas. Con respecto a la última actividad, me parece de gran importancia para comprender que el nodo del que se compone la lista puede albergar cualquier tipo de información, además de que este puede trabajar en conjunto con otras funciones propias de otras estructuras para complementarse.

Las listas ligadas son una herramienta de gran importancia, de la cual se desprenden otras estructuras de datos, y no es de extrañar dadas sus versátiles características, como su flexibilidad o su capacidad de crecer indefinidamente, aunado a la circularidad que puede tener.