



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: TISTA GARCÍA EDGAR

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS I

Grupo: 1

No de Práctica(s): 7) Estructuras de datos lineales: Lista Simple y Lista Circular

Integrante(s): MURRIETA VILLEGAS ALFONSO

Semestre: 2018 - 2

Fecha de entrega: 9 DE ABRIL DE 2018

Observaciones:

CALIFICACIÓN: _____

Estructuras de datos lineales: Lista Simple y Lista Circular

1] Introducción

A lo largo de prácticas anteriores hemos conocido estructuras de datos como son la pila y cola (Con sus respectivas variantes), sin embargo, uno de los detalles particulares es que dentro de cada uno de estos elementos solamente hemos considerado a las listas como simples arreglos.

Las LISTAS son uno de los muchos tipos de estructura de datos lineal y dinámica. Las listas se caracterizan como estructuras lineales debido a que cada elemento tiene un único predecesor y sucesor, por otro lado, se consideran dinámicas porque su tamaño o dimensión no es fijo y se puede definir conforme se requiera.

Las principales operaciones básicas dentro de una lista son BUSCAR, INSERTAR Y ELIMINAR.

2] Objetivo de la práctica

- 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.

3] Desarrollo

3.0] Antecedente y Nodo

Anteriormente se mencionó que las listas son un tipo de estructuras las cuales se tienen como una alternativa para suplir a los arreglos. Pero antes de explicar cada una de las listas que se presentarán dentro de esta práctica, hay que destacar uno de los elementos primordiales de las listas.

En las listas ligadas independientemente del tipo que sea, se componen de una unidad básica conocida como NODO, un nodo se caracteriza por tener un valor cualquier conocido como o utilizado con INFO y además tienen una referencia la cual se usa para enlazar hacia otro nodo para esto utilizamos NEXT. Otro aspecto relevante de los nodos es que se caracterizan por tener posiciones de memoria aleatorias, por ello es que se usan las referencias para encontrarse entre sí (Autoreferenciarse).

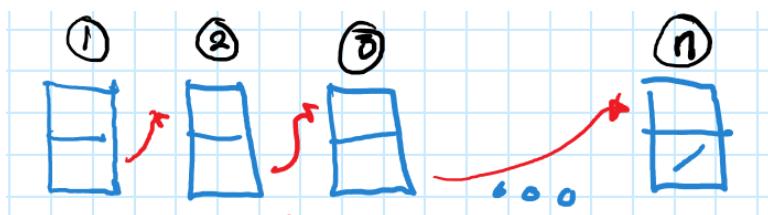


Imagen 1: Nodos enlazados entre sí a través de su segundo elemento que es la referencia o NEXT.

3.1] Análisis de la teoría

LISTA SIMPLE

Una lista simple está constituida por un conjunto de nodos alineados de manera lineal (uno después de otro) y unidos entre sí mediante una referencia (NEXT), a diferencia de un arreglo el cual también es un conjunto de nodos alineados de manera lineal, en las listas simples el orden está determinado por una REFERENCIA, no por un índice, y el tamaño no es fijo.

Como se dijo en el apartado de antecedentes y nodo, la unidad básica de una lista simple es un elemento o nodo. Cada elemento de la lista es un objeto que contiene la información que se desea almacenar, así como una referencia (A través del uso de NEXT) al siguiente elemento también conocido como el sucesor.

NOTA: Un aspecto importante que se verá a continuación es que dentro de las listas existen 2 nodos relevantes que es el caso de HEAD que es el primer nodo dentro de una lista y el caso de TAIL que es el caso del último elemento, sin embargo, comparando la teoría dada en clase con respecto al apartado del manual de prácticas, podemos notar que en el último caso no se hace mención de un nodo llamado Tail y esto es debido a que realmente en el uso de listas lo que nos es más relevantes es el conocer el nodo inicial o HEAD.

Por último, para que realmente se pueda llevar a cabo la naturaleza de este tipo de estructura se necesitan de 3 tipos de funciones u operaciones BUSCAR, INSERTAR Y BORRAR, las cuales serán mencionadas a continuación.

BUSCAR

Una lista simple con elementos puede contener de 1 a n elementos, en tal caso, la referencia al inicio (HEAD) apunta al primer elemento de la lista. Es posible recorrer la lista a través de la referencia (NEXT) de cada nodo hasta llegar al que apunta a nulo, el cuál será el último elemento.

Para poder buscar dentro de esta lista se debe buscar el primer elemento que coincida con la **llave K** dentro de la lista L, a través de una búsqueda lineal simple, regresando un apuntador a dicho elemento si éste se encuentra en la lista o nulo en caso contrario.

NOTA: En caso de la llave k podría verse como un nodo temporal el cual sirve para ir comparando el nodo que se quiere encontrar con los que componen a la lista.

Otro aspecto fundamental es que dentro de las listas la función buscar se caracteriza por tener 3 variantes en la operación búsqueda.

- 1) El primero es que solamente se puede decirnos si existe el elemento o no.
- 2) La segunda variante nos menciona si existe o no el elemento a buscar, pero además nos menciona donde se encuentra.
- 3) Por último, nos puede decir cuántas veces existe el elemento a buscar dentro de la lista, pues como es obvio en la lista puede no solamente existir un solo nodo con la misma información (Apartado de INFO)

NOTA: Como es obvio, dentro de una lista simple vacía o sea que no contiene elementos, la referencia al inicio de la misma (head) apunta a nulo, por lo tanto, en una lista vacía no es posible buscar elementos.

INSERTAR

La función insertar se puede dar en dos casos, el primero es cuando la lista simple se encuentra vacía y el segundo caso es cuando la lista tiene elementos.

Cuando la lista se encuentra vacía, se inserta un nuevo elemento en una lista y la referencia de este está al inicio de la lista (HEAD).

Pero cuando se inserta un nodo en una lista simple con elementos, la referencia del nuevo nodo (NEXT) apunta al mismo nodo al que apunta el inicio de la lista (HEAD) y ahora HEAD apunta al nuevo nodo.

Dicho de otra forma, lo que se necesita para insertar un elemento dentro de una lista con nodos es saber primero el nodo que se encuentra en la posición donde se quiere insertar el nuevo nodo, posteriormente, referenciar el nodo creado al nodo que se encuentra en la posición a insertar, después el antecesor o el nodo anterior referenciarlo al nuevo nodo (Véase de manera visual en la imagen 2).

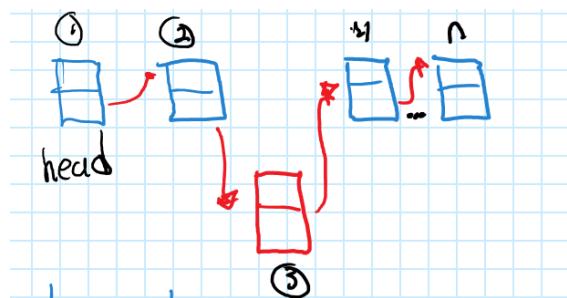


Imagen 2: Diagrama muestra de cómo se vería un nodo que se ha insertado en una lista con elementos.

BORRAR

Para eliminar un nodo en una lista simple con elementos, primero se debe buscar el elemento a eliminar, una vez encontrado el nodo en la lista, se deben mover las referencias de la estructura de tal manera de que el antecesor del nodo a eliminar apunte al sucesor del mismo.

Pero para poder eliminar un elemento de la lista primero es necesario saber la ubicación del nodo a eliminar, por lo tanto, primero se debe realizar una búsqueda del elemento.

NOTA: Recordemos que, debido a los casos de la búsqueda de elementos dentro de la lista, para borrar un elemento podrían existir también variantes ya que podría ser que solamente se elimine el primer elemento (Ejemplo que solamente se borre el número 3 de la lista), sin embargo, podría existir nuevamente este elemento más adelante lo cual haría que se hicieran varios borrados de un mismo elemento.

LISTA CIRCULAR

A diferencia de la lista simple, la lista circular es una lista simplemente ligada “modificada”, ya que, a diferencia de la lista simple, ésta en su apuntador del elemento que se encuentra al final de la lista (TAIL) apunta al primer elemento de la lista (HEAD). Dicho de otra forma, este tipo de estructura se caracteriza de que todos los elementos dentro de esta siempre tienen un sucesor y un antecesor.

Por último, al igual que las listas simples, para que realmente se pueda llevar a cabo la naturaleza de este tipo de estructura se necesitan de 3 tipos de funciones u operaciones que son: BUSCAR, INSERTAR Y BORRAR, las cuales serán mencionadas a continuación.

NOTA: Una de las mayores diferencias entre la lista simple y la lista circular es que la lista circular considera o tiene una variable más y es la **dimensión** o **tamaño** de lista. Es se debe a su naturaleza ya que en las listas circulares si no se considerara la dimensión de estas, al momento de buscar o determinar posiciones no sería posible.

BUSCAR

Una lista circular con elementos puede contener n cantidad elementos, debido a esto, la referencia al inicio (HEAD) apunta al primer elemento de la lista y la referencia a NEXT del último elemento apunta al primer elemento.

Es posible recorrer la lista a través de la referencia (NEXT) de cada nodo, sin embargo, a diferencia de la lista simple se debe tener en cuenta el número de elementos de la lista, ya que el último elemento apunta al inicio de la estructura y, por tanto, se puede recorrer de manera infinita.

INSERTAR

En el caso de la función u operación insertar debido al tipo de estructura se pueden tener distintos casos de uso.

En el caso de que la lista se encuentre vacía es posible insertar elementos, en este caso se inserta un nuevo elemento en una lista circular vacía la referencia al inicio de la lista (HEAD) apunta al nodo insertado y la referencia a NEXT del nodo apunta a sí mismo.

En el caso de que la lista tenga elementos, la referencia del nuevo nodo (NEXT) apunta al mismo nodo al que apunta el inicio de la lista (HEAD) y ahora HEAD apunta al nuevo nodo. Así mismo, el último nodo de la estructura (TAIL) apunta al primer elemento.

BORRAR

Para eliminar un nodo en una lista circular con elementos, primero se debe usar la operación BUSCAR para poder encontrar el elemento a eliminar, una vez encontrado el nodo en la lista, se deben mover las referencias de la estructura de tal manera de que el antecesor del nodo a eliminar apunte al sucesor del mismo. Dicho lo anterior, para poder elementos es necesario saber la ubicación del nodo a eliminar, por lo tanto, primero se debe realizar una búsqueda del elemento

NOTA: Como es obvio en una lista circular vacía no es posible eliminar, debido a que esta estructura no contiene elementos.

3.2] Análisis de los problemas (Práctica)

EJERCICIO 0

Antes de empezar con los ejercicios propuestos por el profesor fue necesario conocer a fondo como es que funcionan las bibliotecas proporcionadas para la práctica. Para ello lo primero que era necesario fue agregarle funciones básicas a las bibliotecas de lista simple y lista circular como es el caso de la función buscar.

Sin embargo, un gran problema surgió al momento de realizar un programa para poder checar las bibliotecas y es que al momento de utilizar la biblioteca de `listacirc` simplemente ocurría un ciclo sin fin debido a que `current2` nunca podía ser nulo (NULL) esto se debía a la naturaleza de la lista circular, es por ello que al ejecutar el programa (ejercicio0) solamente se imprimía una vez la lista.

Para ello lo primero que se tuvo que hacer fue analizar y modificar la parte que ocasionaba este pequeño error, a continuación, se muestra el pedazo de código modificado para poder imprimir las lista cuantas veces quiera el usuario (Véase en imagen 3).

```
int posicion=lista->tamano;

/*
Nodo *current2 =current->next;
while (current2->next != NULL) {
    current = current->next;
}
*/
|
while (posicion != 2) {
    current = current->next;
    posicion--;
}
free(current->next);
current->next = lista->head;
```

Imagen 3: Parte de código de la biblioteca `listacirc`, de verde las líneas comentadas y de azul las líneas agregadas en sustitución de las líneas verdes.

Lo primero que se hizo fue omitir `current2` debido a que ocasionaba problemas al momento de usarlo, para ello se decidió usar `current1` el cual a través del uso de la posición y al meterlo dentro de un ciclo `while` (Se condicionó usando como parámetro la posición) podíamos determinar cómo borrar el último elemento de la lista.

Como resultado final obtuvimos una salida correcta y funcional donde las funciones básicas de la biblioteca ahora si eran funcionales (Véase en la imagen 4).

```
C:\Users\pon_c\OneDrive...
Los elementos de la lista son:
10
20
30
40
=====
El elemento actual es 10
Mostrar siguiente [Presione 1]
Salir [Presione 2]
1
=====
El elemento actual es 20
Mostrar siguiente [Presione 1]
Salir [Presione 2]
```

Imagen 4: Salida del ejercicio 0, podemos ver la lista impresa, el elemento donde se encuentra.

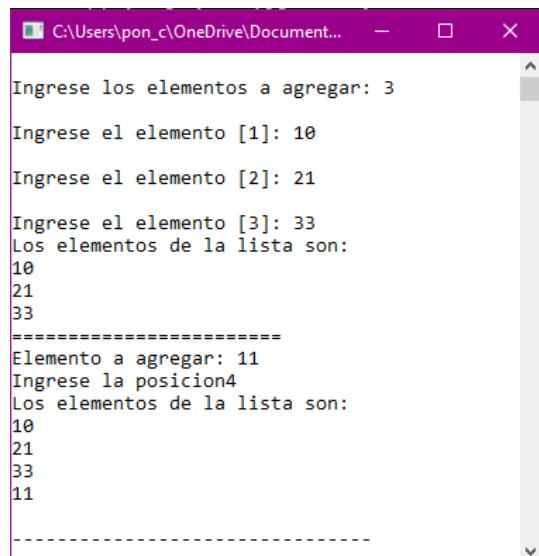
NOTA: También se tuvieron que modificar y agregar algunas funciones para poder emplear la biblioteca en una mayor cantidad de casos. (Fue el caso de la modificación que sufrió la función **borrarPrimero** y agregar la función **buscar2**).

EJERCICIO I

Para un mejor análisis de esta práctica se decidió partir el ejercicio 1 en dos apartados principales, en el primer caso lo que se hizo fue verificar las funciones mediante el uso de un programa, y en el segundo caso se agregó una de las funciones básicas de las listas que es eliminar elementos, pero en este caso no solamente se debía borrar el primer o último elemento sino el elemento que se quisiera o dicho de otra forma el *i*-ésimo.

a) Comprobación de la biblioteca lista.h y función “eliminar *i*-ésimo”

Para hacer este apartado simplemente se usó dentro del programa las funciones que se pedían que eran el caso de **agregar i-ésimo** y **búsqueda** para ello se usaron dos variables, la primera donde se iba a guardar la cantidad de elementos de la lista y la segunda que era el elemento a agregar dentro de la lista, a continuación, en la imagen 5 se puede apreciar la implementación de las dos funciones anteriores.



```
C:\Users\pon_c\OneDrive\Document... - X

Ingrese los elementos a agregar: 3
Ingrese el elemento [1]: 10
Ingrese el elemento [2]: 21
Ingrese el elemento [3]: 33
Los elementos de la lista son:
10
21
33
-----
Elemento a agregar: 11
Ingrese la posicion4
Los elementos de la lista son:
10
21
33
11
```

Imagen 4: Salida del ejercicio 1 (Primer apartado), al principio se muestra la implementación de la función addFinalLista y al final se aprecia de manera implícita la función addIesimoLista

b) Función búsqueda y función borrar *i*-ésimo

Por otro lado, la naturaleza de las listas nos da la oportunidad de poder eliminar elementos en independientemente de la posición en la que se encuentre, por ello, es necesario agregar dentro de la biblioteca lista.h una función que se encargue de la eliminación de cualquier nodo.

Sin embargo, para llevar a cabo la eliminación de cualquier elemento de una lista como es obvio es necesario el uso de otra función básica de las listas que BUSQUEDA. La función búsqueda como las

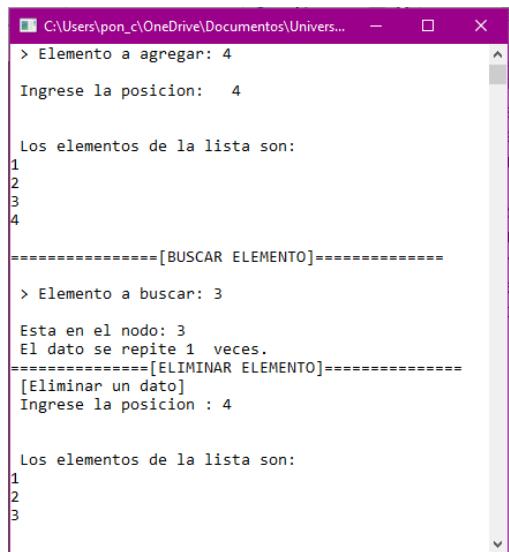
otras funciones básicas necesita como parámetros la lista y un valor extra usado para condicionar la búsqueda.

Como bien sabemos, dentro de la función búsqueda se encuentran 3 variantes de sus resultados:

- 1] Indica si existe el miembro
- 2] Te dice cuántas veces existe el miembro
- 3] Indica donde se encuentra el miembro (Posiciones dentro de la lista)

Pese no era necesario hacer todas las variantes de esta función para el uso de la función borrar i-ésimo, particularmente preferí hacer todas sus variantes de resultados debido a que de esta forma la biblioteca y en específico la función, estarán más completas.

Por otra parte, una vez que se programó la función búsqueda se pudo hacer el algoritmo pertinente de la función eliminar i-ésimo , el cual para poder borrar cualquier miembro primero pasábamos la lista mediante su apuntador (Paso por referencias) debido a que se hará una modificación a esta, además de pasar un valor extra que en este caso es la posición del miembro (nodo) que se quiere eliminar, a continuación se muestra la salida de un programa que hace uso de estas funciones programadas e incluidas en la biblioteca lista.h.



```
C:\Users\pon_c\OneDrive\Documentos\Univers... - □ X
> Elemento a agregar: 4
Ingrese la posicion: 4

Los elementos de la lista son:
1
2
3
4

===== [BUSCAR ELEMENTO] =====

> Elemento a buscar: 3
Esta en el nodo: 3
El dato se repite 1 veces.
===== [ELIMINAR ELEMENTO] =====
[Eliminar un dato]
Ingrese la posicion : 4

Los elementos de la lista son:
1
2
3
```

Imagen 5: Salida del ejercicio 1 (Segundo apartado), al principio se muestra la implementación de la función addI-ésimo, posteriormente se ve el uso de la función BUSQUEDA donde notamos 2 de las posibles respuestas que tiene este tipo de función y por último, se usa la función eliminar i-ésimo.

NOTA:

Dentro de la lista realmente ya teníamos dos funciones de eliminación que son la función de *borrarPrimero* y *borrarUltimo* por ello es que podemos condicionar el uso de estas solo si la posición del nodo a borrar se encuentra en esas posiciones. (Checar código)

c) Función eliminar menores a ...

Lo primero que podemos asumir es que para llevar a cabo esta función es necesario pasar o considerar un elemento que será el valor de comparación con respecto a los valores de los nodos de la lista, el cual además será dado por el usuario. Por lo tanto, para poder borrar los elementos menores a el valor dado por el usuario, primero pasamos la lista mediante su apuntador (Paso por referencias) debido a que haremos una modificación a esta. A continuación, se muestra la salida del programa.

```
C:\Users\pon_c\OneDrive\Documentos\Univers... - □ X
[Eliminar un dato]
Ingrese la posicion : 4

Los elementos de la lista son:
1
2
3
=====
[ELIMINAR MENORES A ]
Se eliminaron los datos menores a:2

Se elimino el nodo: 1
Se borraron 1 dato[s] .

Los elementos de la lista son:
2
3
```

Imagen 6: Salida del ejercicio 1 (Tercer apartado), al principio se muestra la implementación de la función eliminar i-ésimo, posteriormente y como parte principal se muestra la función eliminar menores a, primero indicamos el valor límite de eliminación y después damos tanto que nodo fue eliminado como la cantidad de nodos borrados.

NOTA: Dentro de la función se utilizó algunas variables extras con el propósito de mencionar que nodos fueron eliminados y cuántos nodos fueron borrados.

EJERCICIO II

En el ejercicio dos en primera instancia se pedía usar y comprobar las funciones incluidas en la biblioteca “listacir.h”, posteriormente se le pedía al alumno que mediante esta librería se hiciera modificaciones para poder emplearla para un caso donde los elementos que se iban a usar eran estructuras del tipo de auto. Por lo tanto y para un mejor análisis se dividirá en 2 apartados principales este ejercicio.

a) Comprobación de la biblioteca lista.h y función “eliminar i-ésimo”

Como se mencionó en el análisis de la lista circular las funciones básicas que estas tiene son agregar, borrar y mostrar (Imprimir) donde en el caso de las 2 primeras funciones deben tener la posibilidad de hacerse tanto al inicio como al final, además de dar la oportunidad de hacerlo en cualquier posición de la lista ya que precisamente esa es una de las ventajas que ofrecen las listas con respecto a otros tipos de estructuras como son la pila y cola.

Otro aspecto relevante es que en este caso debido a que estamos hablando de una lista circular no consideramos a un tail como tal, ya que en este caso tenemos otra variable sumamente importante que es el tamaño o dimensión de la lista, recordemos que en este tipo de estructura el último valor tiene como referencia el primero.

Por lo tanto, algunos cambios que se realizaron a la biblioteca “listacir.h” se llevaron principalmente en la función *borrarPrimero* y *borrarultimo*, además de que se realizaron los algoritmos pertinentes para llevar a cabo la función de búsqueda. A continuación, se muestra la salida de un programa que usa las funciones de la biblioteca “listacir(mod).h”.

```
C:\Users\pon_c\OneDrive\Documentos\Universidad\Listas\listacir>=====
===== [INGRESAR DATOS] =====
Ingresar la cantidad de valores en la lista circular: 4
Ingrese el valor en la posición [1]: 1
Ingrese el valor en la posición [2]: 2
Ingrese el valor en la posición [3]: 3
Ingrese el valor en la posición [4]: 4
Los elementos de la lista son:
1
2
3
4
Tamaño actual: 4
Ingrese un nuevo valor al inicio:5
Los elementos de la lista son:
5
1
2
3
4
===== [BORRAR DATOS] =====
```

Imagen 6: Primera parte de la salida del ejercicio 2 (Apartado a)

```
C:\Users\pon_c\OneDrive\Documentos\Universidad\Listas\listacir>=====
===== [Borrar DATOS] =====
Ingresar la cantidad de valores A BORRAR de la lista circular: 1
Los elementos de la lista son:
5
1
2
3
=====
===== [BUSCAR DATO] =====
Valor a buscar: 5
Esta en el nodo: 1
El dato se repite 1 veces.
===== [RECORRER LISTA] =====
El elemento actual es 5
[1] Para ver el siguiente
[2] Salir del programa
1
=====
===== [RECORRER LISTA] =====
El elemento actual es 1
[1] Para ver el siguiente
[2] Salir del programa
```

Imagen 7: Segunda parte de la salida del ejercicio 2 (Apartado a)

En la primera imagen podemos observar el uso de la función *agregarAlFinal* y de la función *imprimirLista*, además del uso de *agregarPrincipioLista*. Por otro lado, en la segunda imagen vemos la implementación de la función *borrarUltimo* y además el uso de la función *búsqueda* la cual fue una de las funciones que fue programada con el fin de completar la biblioteca *listacircular*.

b) Tipo de Dato Abstracto “AUTO”

Para este ejercicio realmente fue necesario dividir parte a parte cada uno de los requisitos que iban a ser necesarios para llevar a cabo un programa que cumpliera con todas las especificaciones solicitadas.

Primero debíamos realizar o elaborar un Tipo de Dato Abstracto que se llamara Auto el cual considerara como datos, características de los autos como la marca, el modelo, la placa, el color y otras más.

Además, se tuvieron que diseñar funciones básicas de las listas para esto simplemente se tomó el apartado anterior y se hicieron los cambios necesarios.

Entre las funciones que fueron adaptadas se encuentra, *imprimirLista*, agregar y eliminar al inicio y al final de la lista. A continuación, se muestra la salida del programa usando estas 3 funciones.

```

C:\Users\pon_c\OneDrive\Documentos\Universid...
===== [INGRESAR AUTOS] =====
Ingrese la cantidad de autos a ingresar: 2
===== [DATOS] =====
Ingrese el los datos del auto en la posición 1:
Marca: nissan
Modelo: versa
Edición(Año de salida del auto): 2018
Matrícula: asdfg
Color: azul
Pasajeros:4
===== [DATOS] =====
Ingrese el los datos del auto en la posición 2:
Marca: toyota
Modelo: prius
Edición(Año de salida del auto): 2019
Matrícula: qwerty
Color: rojo
Pasajeros:5

```

Imagen 8: Primera parte de la salida del ejercicio 2 (Apartado b-c-d), uso de la función agregar al final

```

C:\Users\pon_c\OneDrive\Documentos\Universid...
===== [ELEMENTOS DE LA LISTA] =====
Marca: nissan
Modelo: versa
Edición: 2018
Matrícula: asdfg
Número de pasajeros: 4
Color: azul

Marca: toyota
Modelo: prius
Edición: 2019
Matrícula: qwerty
Número de pasajeros: 5
Color: rojo
===== [OPCIONES] =====
1] Agregar auto
2] Eliminar auto
3] Buscar auto
4] Imprimir lista
5] Salir.
Escoja una de las opciones:

```

Imagen 9: Segunda parte de la salida del ejercicio 2 (Apartado b-c-d), uso de la función imprimir y despliegue del menú principal

NOTA: En el caso de la función imprimir solamente se pasa por valor debido a que no necesita hacerse cambios, sin embargo, para agregar o eliminar elementos se necesitó pasar por referencia (Se utilizó apuntador a la lista)

Por último, se nos pidió hacer tanto la función de hacer una búsqueda de autos dentro de la lista mediante el uso de la marca, en este apartado realmente fue algo difícil hacer el algoritmo de comparación, sin embargo, lo que debemos considerar es que realmente no debemos comparar todo el dato sino uno de los elementos del TDA que en este caso es la marca. Para esto en la función búsqueda simplemente se pasa la lista por valor y de esta forma dentro de esta función hacemos la comparación de cadenas.

NOTA: Recordemos que en el lenguaje C como tal no tenemos cadenas, por ello no podemos comparar directamente los nombres de las marcas.

Para hacer la comparación de las cadenas fue necesario utilizar la biblioteca **string.h** y en específico la función **strcmp()**. A continuación, se muestra la salida del programa del programa donde nos indica si existe o no el auto y además nos da la posición donde se encuentra dentro de la lista y cuantos modelos de ese auto existen.

```

===== [BUSQUEDA] =====
Ingresar marca del AUTO a buscar: nissan
Su opción nissan esta disponible, en la posición: 1
Se encontraron 1 modelos de la marca nissan.
===== [OPCIONES] =====
1] Agregar auto
2] Eliminar auto
3] Buscar auto
4] Imprimir lista
5] Salir.
Escoja una de las opciones:

```

Imagen 10: Tercera parte de la salida del ejercicio 2 (Apartado b-c-d), uso de la función búsqueda.

3.4] Notas y características de las bibliotecas

Para poder comprender con su totalidad a las bibliotecas empleadas a lo largo de esta práctica, tanto en las bibliotecas hechas por el profesor como las creadas se les ha agregado función por función tanto los parámetros que necesitan como las características que cada función tiene. (Véase todas estas características en las bibliotecas correspondientes).

3.4] Relación entre los ejercicios y el tema

Las listas son un tipo de estructura sumamente importante, a lo largo de esta práctica y mediante la abstracción y análisis de distintos problemas se empleó 2 principales estructuras (Listas simples y listas circulares) para poder facilitar y optimizar la manera resolver los problemas propuestos.

La relación de los ejercicios con respecto al tema que estamos viendo que es sobre todo una estructura de datos lineal que es el caso de lista podemos ver sobre todo la amplia relación que existe por ejemplo en el ejercicio 1 en específico en el apartado b y c donde notamos como las características básicas de la lista son las que nos pueden ampliar un sinfín de oportunidades dentro de nuestros programa y es que por ejemplo al usar la función búsqueda podemos realizar otras función como fue el caso de la función borrar_elementos_menores_a. Lo importante de esto es reconocer la naturaleza y las ventajas que nos ofrece cada una de las estructuras que estamos viendo.

Otro caso muy destacable fue el último ejercicio donde al plantearnos un tipo de abstracto llamado carro el cual como datos tenía las características de los autos que el usuario ingresara, pudimos notar que al conocer las funciones básicas de la lista circular, pero sobretodo como es que estas funcionaban nos ayudó y resultó de una manera más cómoda y agradable el poder realizar un programa donde pudiéramos consultar todo hacer de los autos encontrados dentro de esa lista.

4] Conclusiones

Esta práctica fue el primer acercamiento a uno de los conceptos más relevantes de las estructuras de datos, es el caso de las listas, las cuales entre los muchos propósitos que tienen está el sustituir a los arreglos usados dentro de las pilas y colas y es que, este tipo de estructura ofrece una manera muy distinta de manejar los datos a comparación de los arreglos.

Las ventajas que ofrece es sobre todo la enorme libertad que tienen incluso comparándola con otros tipos de estructuras como son las pilas y colas, y es que este tipo de estructura nos permite el agregar y eliminar elementos en cualquier parte de la estructura (De la lista).

Otro aspecto relevante fue como para poder emplear de manera apropiada esta estructura usamos 3 principales operaciones que son BUSCAR, INSERTAR y ELIMINAR donde sobretodo el uso adecuado del paso por referencia y el paso por valor es indispensable para no cometer errores.

Algo a destacar es que las listas son una de las estructuras de datos más utilizadas en las ciencias de la computación. Un ejemplo más que sorprendente es que en cualquier red social, siempre se usan las listas dentro de los *time-line* o caja de contenidos, ya que cada elemento tiene un único sucesor que sería la

siguiente publicación, hasta llegar a la última publicación o elemento dentro de la lista o caja de contenidos. Es más que notorio que el uso de estructura tiene el principal propósito de que se puedan realizar **N** cantidad de publicaciones, siempre insertando por delante (TAIL) y la última publicación (TAIL) no tiene sucesor (NULO).

Por otro lado, también es destacable el uso de las listas circulares y es que este tipo de estructuras se pueden ver por ejemplo en los reproductores de música al momento de terminar las listas pueden tener la opción de que se regresen al primer elemento para nuevamente empezar la lista de reproducción. O también podemos ver estas estructuras cuando varios procesos se están utilizando el mismo recurso durante la misma cantidad de tiempo.

Por último, si quisiera mencionar que la parte que más conflictos me dio fue la búsqueda de un auto a través de su marca y es que realmente la manera en que abordé este apartado fue a través de la comparación de cadenas mediante la función strcmp().

5] Referencias

- 1) Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, McGraw-Hill.
- 2) El lenguaje de programación C. Brian W. Kernighan, Dennis M. Ritchie, segunda edición, USA, Pearson Educación 1991.
- 3) Curso de Programación C / C++. Javier Ceballos, tercera edición. USA, Ra – Ma. Educación 2007