	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Edgar Tista García
Asignatura:	Estructuras de Datos y Algoritmos I
Grupo:	1
No de Práctica(s):	Práctica #7 - #8
Integrante(s):	Santiago Díaz Edwin Jaret
No. de Equipo de cómputo empleado:	Trabajo en Casa
No. de Lista o Brigada:	42
Semestre:	2021-2
Fecha de entrega:	11/07/2021
Observaciones:	

CALIFICACIÓN: _____

Implementación de Listas Ligadas

Objetivos

Objetivo (7): Revisar las definiciones, características, procedimientos y ejemplos de las estructuras Lineales Lista simple y Lista Circular, con la finalidad de que comprender sus estructuras e implementarlas.
Objetivo (8): Revisar las definiciones, características, procedimientos y ejemplos de las estructuras lineales doblemente ligada y lista doblemente circular, con la finalidad de que comprender sus estructuras e implementarlas.

Desarrollo

Los programas propuestos por el profesor de "lista.c" y "listacirc.c" tenían algunos errores, como la ausencia de punto y coma, de las bibliotecas "<**stdio.h**>, <**stdlib.h** >" y ausencia de comillas dobles en ciertas partes.
Los programas de la actividad 1, actividad 2 y actividad 3 se encuentran documentados y utilizan una biblioteca llamada "*acentos.h*" en donde se encuentran variables de tipo **char** para caracteres especial usando el código ASCII, esto es para tener una mejor visualización del programa.

Actividad 1.

Se creó un programa llamado *"menu.c"* en donde se utiliza las operaciones y estructura de la lista las cuales se encuentran en la biblioteca "*lista.c*". Estas son impresas de forma de un menú las cuales, el usuario ingresa el número de instrucción que desea ejecutar, si se ingresa un número distinto de las instrucciones, el programa se finaliza. El menú se imprime de la siguiente manera:

```
        ¦ MENÚ ¦

Estas son las operaciones disponibles:

        1.- Imprimir los valores de los nodos.
        2.- Agregar al principio de la lista un Nodo.
        3.- Agregar al final de la lista un Nodo.
        4.- Agregar en la posición N de la lista un Nodo.
        5.- Borrar el primer Nodo de la lista.
        6.- Borrar el último Nodo de la lista.
        7.- Buscar un elemento
        8.- Eliminar el nodo N-ésimo
        9.- Eliminar los elementos menores de un elemento
        10.- Obtener el valor del primer nodo de la lista.
        11.- Terminar el programa
¿Qué operacion desea realizar?
```

La estructura **Nodo** contiene una variable de tipo **int** con el nombre *val* y otra estructura de tipo **Nodo** con el nombre *next*. La estructura **Lista** contiene un **Nodo** con el nombre *head*.

El programa inicia desde la función *main()* en donde se crea una **lista** con la función `crearLista()`, después se ejecuta el menú, conforme el usuario ingresa el número de instrucción se ejecutará las funciones respectivamente.

Las funciones con las que se trabajan junto con su objetivo son:

- `crearLista()` Crea una lista, el nodo **head** lo iguala a **NULL** y esta lista se retorna.
- `print_list(Lista)` Recibe como parámetro una estructura **Lista**. la función imprime los valores de la lista, pero primero, valida si la lista tiene elementos o no, si no tiene elementos se le notifica al usuario, de lo contrario, a través de un nodo **current** recorre la lista, empezando desde el nodo **head** hasta el último y en cada nodo que se encuentre, se imprime su valor.
- `addPrincipioLista(Lista*, int)` Recibe como parámetro una estructura **Lista** y un valor de tipo **int.** La función crea y agrega al principio de la lista un nodo llamado

new_node, su valor es el valor del parámetro de la función. Al final, este es el nuevo nodo **head**.

- `addFinalLista(Lista*, int)` Recibe como parámetro una estructura **Lista** y un valor de tipo **int.** Se valida si se encuentra vacía o no. Si esta está vacía, se crea un nodo y este se convierte en el último nodo; si no está vacía, con la ayuda de un nodo **current** se obtiene el último nodo de la lista para convertirse en el penúltimo y con otro nodo **nuevoNodo** se convierte en el último de la lista.
- `borrarPrimero(Lista*)` Recibe como parámetro una estructura **Lista** y un valor de tipo **int.** Se valida si la lista se encuentra vacía, si fuera el caso se le notifica al usuario; de lo contrario, se elimina el nodo **head** y el segundo nodo queda como primer nodo y este sería el nuevo nodo **head**.
- `borrarUltimo(Lista*)` Recibe como parámetro una estructura **Lista** y un valor de tipo **int.** Valida si la lista se encuentra vacía, si fuera el caso se le notifica al usuario; de lo contrario, se elimina el último nodo con la ayuda de un nodo **current**, el cual, es el penúltimo nodo.
- `primerElemento(Lista)` Recibe como parámetro una estructura **Lista.** Retorna el primer elemento de la lista.
- `addIesimoLista(Lista*, int, int)` Recibe como parámetro una estructura **Lista** y dos valores de tipo **int,** el primero es el valor a agregar y el segundo la posición. Valida si se encuentra vacía, si fuera el caso, se le pregunta al usuario si desea agregar un nodo o no. Si el usuario desea ingresar uno, se manda a llamar a la función `addPrincipio-Lista`. Si no se encuentra vacía, se valida que la posición exista dentro de la lista y si cumple con esto, se obtiene un nodo anterior a la posición deseada para poder agregar el nuevo valor. Si la posición no existe, se agrega en la última posición.
- `busqueda(Lista, int)` Recibe como parámetro una estructura **Lista** y un valor de tipo **int.** Con la ayuda de un nodo **temp** se recorre cada elemento de la lista buscando el elemento solicitado por el usuario y se lleva un conteo de cuántas veces se encontró el elemento en la lista.
- `borrarIesimoLista(Lista*, int)` Recibe como parámetro una estructura **Lista** y un valor de tipo **int.** Se valida si la lista se encuentra vacía o no, si está vacía, se le notifica al usuario; si no está vacía, se valida cuántos elementos contiene la lista, debido a que si hay solo 1 elemento, se elimina y no se tiene que realizar alguna operación de conexión. Si hubiera más elementos, se obtiene el nodo anterior al deseado a eliminar, para que este direccione a la posición+1 y no se pierda la lista.
- `borrarMenoresLista(Lista*, int)` Recibe como parámetro una estructura **Lista** y un valor de tipo **int.** Recorre toda la lista con la ayuda de un nodo **temp** y si el valor del nodo actual es menor al valor ingresado, se manda a llamar a la función `borrarIesimo-Lista()` para eliminar esa posición del nodo. Si este es igual o mayor al valor ingresado, se pasa al siguiente nodo.

A continuación, se presenta la ejecución del programa indicando que funciones se realizan y los resultados impresos en la pantalla.

```
addPrincipioLista(4)
addPrincipioLista(3)
addPrincipioLista(2)
addFinalLista(5)
addFinalLista(6)
addFinalLista(7)
print_lista()
```

```
¿Qué operacion desea realizar?
2
¿Qué elemento desea agregar?
2
```

```
¿Qué operacion desea realizar?
3
¿Qué elemento desea agregar?
7
```

```
¿Qué operacion desea realizar?
1
Los elementos de la lista son:
2
3
4
5
6
7
```

```
addIesimoLista(1,1)
addIesimoLista(4,4)
borrarPrimero()
borrarUltimo()
borrarPrimero()
borrarUltimo()
print_lista()
```

```
¿Qué operacion desea realizar?
4
¿Qué elemento desea agregar?
4
¿En qué posición?
4
```

```
¿Qué operacion desea realizar?
1
Los elementos de la lista son:
3
4
4
5
```

```
busqueda(4)
busqueda(7)
```

```
¿Qué operacion desea realizar?
7
¿Cuál es el elemento que desea buscar?
4
El elemento 4 se encuentra 2 veces
```

```
¿Qué operacion desea realizar?
7
¿Cuál es el elemento que desea buscar?
7
El elemento 7 se encuentra 0 veces
```

```
borrarIesimoLista(2)
print_lista()
```

```
¿Qué operacion desea realizar?
8
Qué posición desea eliminar?
2
```

```
¿Qué operacion desea realizar?
1
Los elementos de la lista son:
3
4
5
```

```
addPrincipioLista(7)
addPrincipioLista(8)
addPrincipioLista(9)
addPrincipioLista(10)
addPrincipioLista(4)
addPrincipioLista(3)
borrarMenoresLista(8)
print_lista()
```

```
¿Qué operacion desea realizar?
9
¿Cuál es el elemento mayor?
8
```

```
Imprimimos la lista con valores menores:
3       4       7       3       4       5
Se eliminaron #6 elementos
```

```
¿Qué operacion desea realizar?
1
Los elementos de la lista son:
10
9
8
```

```
primerElemento()
```

```
¿Qué operacion desea realizar?
10
Primer elemento: 10
```

```
¿Qué operacion desea realizar?
11
Adiós
```

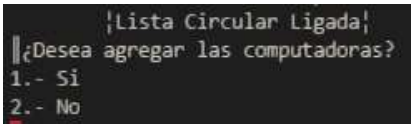
Para terminar el programa, se ingresa un número igual o mayor a 11.

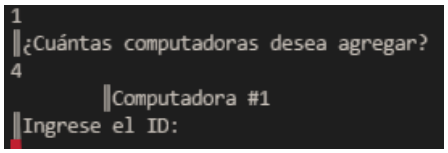
Actividad 2.

Se trabaja en el programa "*listacirc.c*" el cual, con los fundamentos de una lista circular simple, se crea dicha lista de tipo de estructura y un nodo de tipo estructura.

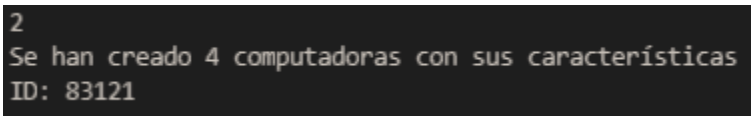
La estructura **NodoC** contiene una variable de tipo estructura **Computadora** llamada *val* y otra estructura de tipo **Nodo** con el nombre *next*. La estructura **ListaC** contiene un **Nodo** con el nombre *head* y una variable de tipo **int** con el nombre de *tamano*. La estructura **Computadora** contiene una variable *id* de tipo **int** y 5 variables de tipo **char** con los nombres de *marca, tipo, procesador, memoria, sistemaOperativo*.

El programa inicia desde la función *main()* el cual, crea una **lista** con la función `crearListaC()`, después, se le pregunta al usuario si desea crear nuevos elementos de tipo **Computadora** o trabajar con los 4 elementos previamente creados, esto fue incluido para fines más prácticos, como se muestra de la siguiente manera.



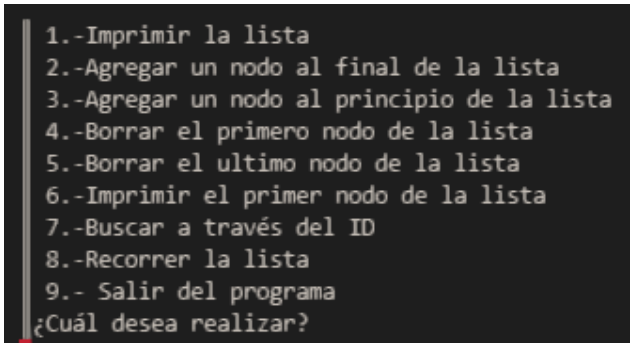


Se le solicitará al usuario que ingrese la información necesaria para llenar la estructura **Computadora**.



Se imprime las computadoras que fueron creadas

Independientemente si el usuario decide agregar o no elementos de tipo **Computadora**, se imprime el mismo menú el cual contiene las siguientes instrucciones:



Las funciones con las que se trabajan junto con sus objetivos son:

- `crearListaC()`, crea una variable de tipo **ListaC** llamada **listaC** con su valor de nodo **head** en **NULL** y su tamaño de lista en **0**. Se retorna la **listaC**.
- `print_listC(Lista)` imprime los elementos de la lista siempre y cuando contenga elementos, si no contiene, se le notifica al usuario que la lista se encuentra vacía. Para imprimir los elementos, se crea un nodo **temp** el cual, recorre la lista desde **head** hasta el último nodo.
- `addFinalListC(Lista*, Computadora)` agrega al final de la lista un nuevo elemento de tipo computadora. Se necesita crear un nodo **current** para llegar al último nodo. Después, se crea otro nodo **nuevoNodoC** el cual, será el último, al final, se aumenta el tamaño de la lista.
- `addPrincipioListaC(ListaC*, Computadora)` Recibe como parámetro una estructura de tipo Computadora y la lista. Agrega al principio de la lista un nuevo nodo con el elemento de tipo computadora, al final, se aumenta el tamaño.
- `borrarPrimeroC(ListaC*)` Recibe como parámetro la lista circular. Valida si esta se encuentra vacía, si fuera el caso, notifica al usuario; de lo contrario utilizamos el último nodo para que direccione al nodo número 2, para sacar al **head**. El nodo 2 se convierte en el nodo 1 y en el nuevo **head**.
- `borrarUltimoC(ListaC*)` Recibe como parámetro la lista circular. Para borrar el último nodo se necesita 2 nodos, uno se posicionará en el último nodo y se igualará a **NULL** y el otro se posicionará en el penúltimo para después, direccionarlo al nodo **head**.

- `primerElementoC(ListaC)` Recibe como parámetro la lista circular. Imprime el primer valor de la lista, el cual, se encuentra en el nodo **head**.
- `BuscarCircC(ListaC, int)` Recibe como parámetro la lista circular y un valor de tipo **int**. Se recorre la lista a través de un nodo para buscar el ID de una computadora y si el valor del ID nodo actual es el mismo ID ingresado por el usuario, se imprime la información de la computadora; de lo contrario, se le notifica al usuario que no se encontró el ID
- `recorrerLista(ListaC)` Recibe como parámetro la lista circular. A través de las teclas 'a', 'd', se recorre la lista y se muestra hacia qué dirección se está recorriendo y el ID de la computadora. La tecla 'g' muestra la información de la computadora actual y con la tecla 'j' se sale de esta función.

A continuación, se presenta la ejecución del programa indicando que funciones se realizan y los resultados impresos en la pantalla. Para este caso, no se decidió agregar las computadoras desde el inicio del programa, por lo tanto, se trabaja con las computadoras previamente creadas.

```
addFinalListaC(computadora)
addPrincipioListaC(computadora)
```

```
¿Cuál desea realizar?
2
        Computadora #1
Ingrese el ID:
8888
Ingrese la marca:
Lenvo
Ingrese la cantidad de memoria:
64 GB RAM
Ingrese el procesador:
Intel Corei 9
Ingrese el sistema operativo:
Windows 11
Ingrese el tipo de computadora:
Laptop
```

```
¿Cuál desea realizar?
3
        Computadora #1
Ingrese el ID:
3333
Ingrese la marca:
Mac
Ingrese la cantidad de memoria:
16 GB RAM
Ingrese el procesador:
M1
Ingrese el sistema operativo:
Mac OS
Ingrese el tipo de computadora:
Desktop
```

```
print_listC()
```

```
1
        Los elementos de la lista son:
ID: 3333
Marca: Mac
Memoria: 16 GB RAM
Procesador: M1
Sistema Operativo: Mac OS
Tipo de Computadora: Desktop
```

```
ID: 83121
Marca: DELL
Memoria: 16GB RAM
Procesador: Intel CoreI9
Sistema Operativo: Windows
Tipo de Computadora: Laptop
```

```
ID: 9120823
Marca: Lenovo
Memoria: 64GB RAM
Procesador: Intel CoreI7
Sistema Operativo: Windows
Tipo de Computadora: Desktop
```

```
ID: 357320
Marca: Alienware
Memoria: 24GB RAM
Procesador: Ryzen 9 3950X
Sistema Operativo: Windows
Tipo de Computadora: Laptop
```

```
ID: 729012
Marca: Apple
Memoria: 32GB RAM
Procesador: M1
Sistema Operativo: MAC OS X
Tipo de Computadora: Desktop
```

```
ID: 8888
Marca: Lenvo
Memoria: 64 GB RAM
Procesador: Intel Corei 9
Sistema Operativo: Windows 11
Tipo de Computadora: Laptop

Tamaño: 6
```

```
borrarPrimeroC()
borrarPrimeroC()

primerElementoC()
```

```
6
ID: 9120823
Marca: Lenovo
Memoria: 64GB RAM
Procesador: Intel CoreI7
Sistema Operativo: Windows
Tipo de Computadora: Desktop
```

```
BuscarCirc(729012)
BuscarCirc(555732)
```

```
7
¿Cuál es el ID que desea buscar?:
729012

ID: 729012
Marca: Apple
Memoria: 32GB RAM
Procesador: M1
Sistema Operativo: MAC OS X
Tipo de Computadora: Desktop

En la posición 3
```

```
7
¿Cuál es el ID que desea buscar?:
555732

No se encontró el ID :(
```

```
recorrerLista()
```

```
g

Ingrese 'a' para moverse al nodo de la izquierda
Ingrese 'd' para moverse al nodo de la derecha
Ingrese 'g' para ver la información del nodo
Ingrese 'j' para salir de la ejecución
```

```
d

Se mueve a la Derecha
ID: 357320
MARCA: Alienware

d

Se mueve a la Derecha
ID: 729012
MARCA: Apple

d

Se mueve a la Derecha
ID: 8888
MARCA: Lenvo

d

Se mueve a la Derecha
ID: 9120823
MARCA: Lenovo
```

```
g

Se Imprime la información
ID: 9120823
Marca: Lenovo
Memoria: 64GB RAM
Procesador: Intel CoreI7
Sistema Operativo: Windows
Tipo de Computadora: Desktop

a

Se mueve a la izquierda
ID: 8888
MARCA: Lenvo

a

Se mueve a la izquierda
ID: 729012
MARCA: Apple
```

```
g

Se Imprime la información
ID: 729012
Marca: Apple
Memoria: 32GB RAM
Procesador: M1
Sistema Operativo: MAC OS X
Tipo de Computadora: Desktop

j

Decidió salir de la operación
```

Actividad 3.

Para trabajar con la lista ligada circular doble, se requirió tener conocimiento previo de esta estructura lineal para realizar la actividad, una vez cumplido esta parte, se creó un programa con el nombre "*listacircdoble.c*" en donde las operaciones con las que se trabaja se van a imprimir de manera de un menú para que el usuario pueda elegir que se va a realizar. El menú se imprime siempre y cuando la instrucción ingresada por el usuario sea parte del menú, esto se ejecuta con un ciclo Do-While().

La estructura **NodoDoble** contiene una variable de tipo **int** con el nombre *val*, una estructura de tipo **NodoDoble** con el nombre *next* y otra estructura de tipo **NodoDoble** con el nombre *prev*. La estructura **ListaDoble** contiene un **NodoDoble** con el nombre *head* y una variable de tipo **int** con el nombre *tamano*.

Las instrucciones junto con sus funciones son:

```
¦Menú de Lista Ligada Doble¦

1.-Agregar un elemento al principio
2.-Agregar un elemento al final
3.-Agregar un elemento en la posicion N
4.-Eliminar el primer elemento
5.-Eliminar el último elemento
6.-Imprimir la lista
¿Con cuál desea trabajar?
```

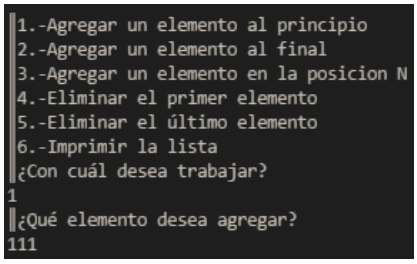
```
agregarPrincipioNodoDoble(ListaDoble*, int);
agregarFinalNodoDoble(ListaDoble*, int);
agregarIesimoNodoDoble(ListaDoble*, int, int);
eliminarInicioNodoDoble(ListaDoble*);
eliminarFinalNodoDoble(ListaDoble*);
imprimirListaDoble(ListaDoble);

validarPosicion(ListaDoble, int);
siEstaVacio(ListaDoble*, int );
```

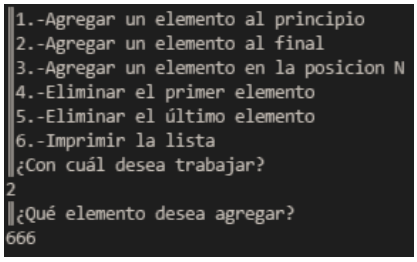
Lo que realiza cada función es lo siguiente:

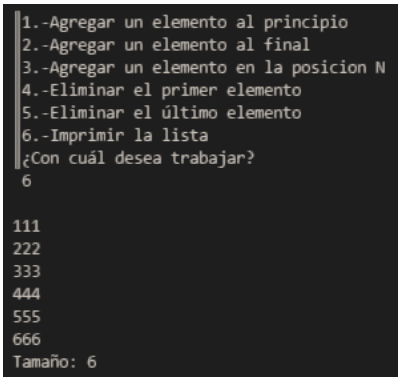
- La función `agregarPrincipioNodoDoble()` recibe como parámetro la estructura **ListaDoble** como apuntador para que pueda ser modificado y una dato de tipo **int**, el cual es el valor del nuevo nodo a agregar. Se valida si la lista se encuentra vacía y si fuera el caso, se manda a llamar la función `validarPosicion()` y si no se encuentra vacía, se crea un **nuevo** nodo y se agrega al principio de la lista, al final, el nodo **head** será el **nuevo** nodo agregado.
- La función `agregarFinalNodoDoble()` recibe como parámetro la estructura **ListaDoble** como apuntador y un dato de tipo **int**. Se valida si la lista se encuentra vacía, si fuera el caso se manda a llamar la función `validarPosicion()` y si no se encuentra vacía, un **nuevo** nodo se crea y se agrega al final de la lista.
- La función `agregarIesimoNodoDoble()` recibe como parámetro la estructura **ListaDoble** como apuntador y dos datos de tipo **int**, uno para el valor del nodo y otro para la posición de la lista donde desea agregar el nuevo nodo.
- La función `eliminarInicioNodoDoble()` recibe como parámetro la estructura **ListaDoble** como apuntador. Este eliminará el nodo **head**, reduciendo el tamaño de la lista y se le asigna al segundo nodo como el **nuevo head**.
- La función `eliminarFinalNodoDoble()` recibe como parámetro la estructura **ListaDoble** como apuntador. Este eliminará el último nodo de la lista para esto, es necesario obtener el penúltimo nodo y a través de ese nodo y el **head**, se cambian las direcciones de **.next** y **.prev** para no apuntar al último nodo. Al final, se le asigna un valor de **NULL.**
- La función `imprimirListaDoble()` recibe como parámetro la estructura **ListaDoble**. Se crea un nodo **temp** el cual es igualado al nodo **head**, para recorrer cada nodo de la lista e imprimir los valores de cada nodo.
- La función `siEstaVacio()` recibe como parámetro la **ListaDoble** y un valor de tipo **int**. Esto es cuando la lista se encuentra sin ningún nodo y se quiere agregar uno, el cual, se inicializa el nodo **head** con el valor que recibe como parámetro la función y su **.next** y **.prev**, apuntan al mismo nodo **head**.
- La función `validarPosicion()` es para validar si la posición en donde desea agregar un elemento existe en la lista, este regresa un **0** si este no existe y **1** si existe. Esta función se utiliza con la función `agregarIesimoNodoDoble()`.
- `crearListaDoble()` La función crea una **ListaDoble** con un valor **NULL** a **head** y un valor de **0** a **tamano**. Regresa la lista creada.

Un ejemplo de como funciona el programa:

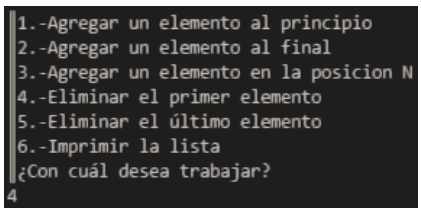
```
agregarPrincipioNodoDoble(333)
agregarPrincipioNodoDoble(222)
agregarPrincipioNodoDoble(111)
agregarFinalNodoDoble(444)
agregarFinalNodoDoble(555)
agregarFinalNodoDoble(666)
imprimirListaDoble()
```

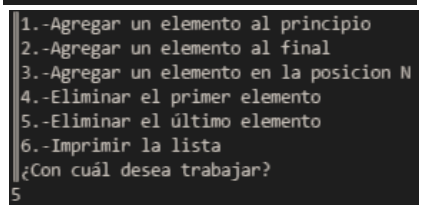


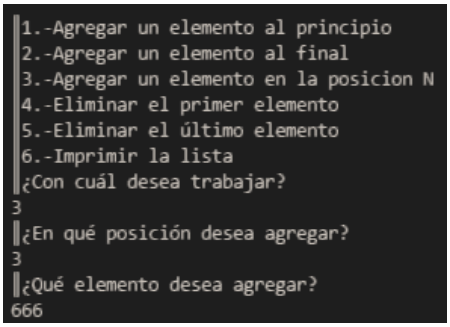


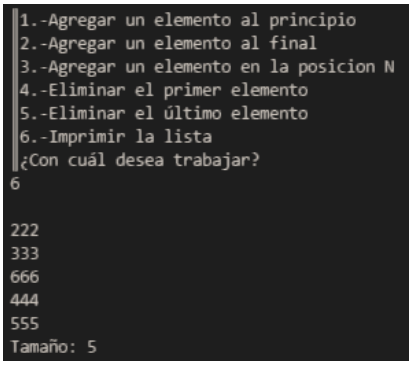



```
eliminarInicioNodoDoble()
eliminarFinalNodoDoble()


agregarIesimoNodoDoble(3, 666)

imprimirListaDoble()
```









El programa cumple con los puntos requeridos.

Las ventajas de utilizar la lista ligada circular doble es que hay un mejor acceso a los nodos, hay un mayor flujo y flexibilidad en el programa y se redujo el tamaño del programa. La desventaja de trabajar es que se puede confundir más en la aplicación de la lista en el código.

Conclusiones.

La práctica contiene la explicación de los 3 ejercicios propuestos por el profesor y cumplido en su totalidad. Cada ejercicio contiene al menos un programa el cual se encuentra documentado y en cada ejercicio se cumplió los puntos solicitados por el profesor.

Los objetivos de la práctica fueron cumplidos al utilizar las estructuras lineales de lista simple, lista circular simple y lista circular doble.

La práctica se me complicó mucho pues llegaba momentos en el que el código terminaba de ejecutarse de manera imprevista, los valores de los nodos no eran los correctos, la listas no se ligaban, los ciclos utilizados se iban al infinito. Aprendí mucho al realizar esta práctica y con lo que más me llevo es que es fundamental reservar un espacio en la memoria de una estructura nodo ya que, al no reservar me causaba más problemas y eso me atrasó mucho en el avance.

Con lo aprendido, se puede construir programas y ser aplicados en la vida cotidiana y esto será la base para otras estructuras como grafos y árboles.