

UNIVERSIDAD DE COSTA RICA
Escuela de Ingeniería Eléctrica
IE0117 – Programación bajo plataformas abiertas

Laboratorio 7

Yeison Rodríguez Pacheco, B56074

1/06/18

Índice

1. Introducción	4
2. Nota teórica	4
3. Análisis de resultados	5
3.1. Main y compilación	5
3.1.1. Compilación utilizando Make	5
3.1.2. Ejecutando el programa y explicación del main	6
3.2. Funciones	7
3.2.1. pos_t* createList(int first_value)	7
3.2.2. void printList(pos_t* head)	8
3.2.3. int push_back(pos_t* head, int new_value)	9
3.2.4. int push_front(pos_t** head, int new_value)	10
3.2.5. int pop_front(pos_t **head)	11
3.2.6. int pop_back(pos_t* head)	12
3.2.7. int insertElement(pos_t** head, int pos, int new_value)	13
3.2.8. int removeElement(pos_t** head, int pos)	14
3.2.9. int getElement(pos_t* head, int index, int* valid)	15
3.2.10. void sort(pos_t* head, char dir)	16
3.2.11. pos_t* readList(const char* filePath)	18
3.2.12. void writeList(pos_t* head, const char* filePath)	19
3.2.13. int freeList(pos_t* head)	20
4. Conclusiones y recomendaciones	21

Índice de figuras

1. Estructura de una lista enlazada. Tomado de [1]	5
2. Directorio make antes de compilar con comando make	6
3. Compilando con Make el programa	6
4. Menú inicial del programa	7
5. Menú principal del programa.	7
6. Utilizando función createList	8
7. Utilizando función printList	9
8. Utilizando función pushback	10
9. Utilizando función pushfront	11
10. Utilizando función pop_front	12
11. Utilizando función pop_back	13
12. Utilizando función insert element	14
13. Utilizando función remove element	15
14. Utilizando función get element	16
15. Utilizando función sort de manera ascendente	17
16. Utilizando función sort de manera descendente	18

17.	Utilizando función read list	19
18.	Utilizando función writelist en el menú	20
19.	Verificando función writelist	20
20.	Verificando función freelist	21

1. Introducción

A la hora de aprender el arte de la programación es útil realizar todo desde cero ya que se necesita aprender a manejar los conocimientos base, pero llega un punto en la vida de un programador que no es eficiente tener que programar todo lo que se va a utilizar, por lo que existe el concepto de biblioteca que es básicamente un conjunto de funciones realizadas por algún desarrollador que resuelven problemas específicos, un ejemplo de esto es la biblioteca `math` en C que contiene funciones para resolver raíces cuadradas, cosenos, entre otros. Así que si se necesita resolver una raíz cuadrada para algún proyecto, lo más recomendable es utilizar una biblioteca que facilite el problema.

En este laboratorio se creará una biblioteca que permite manejar listas enlazadas por medio de un conjunto de funciones, por lo que si en algún momento se necesita trabajar con este tipo de organización de datos ya se tendría casi todo el trabajo listo al incluir esta biblioteca. Una lista enlazada consiste en un conjunto de datos que están enlazados el uno con el otro en una cadena, si se pierde un eslabón de la misma es posible que deje de funcionar, por lo que se deben trabajar con cuidado. La lista enlazada cumple un funcionamiento similar al del arreglo con la principal diferencia de que todos los datos de la lista se almacenan en distintos espacios de memoria que no precisamente son lineales como ocurre en un arreglo. Existen muchos tipos de listas enlazadas, algunas contienen la dirección del elemento anterior y el posterior, en otras el elemento final tiene la dirección del inicial por lo que se cuenta con un ciclo de memoria.

Para la creación de dicha biblioteca se van a utilizar todos los conocimientos con respecto al lenguaje de programación C adquiridos en el curso, como lo son funciones, estructuras, punteros, memoria dinámica, lectura y escritura de archivos binarios y de texto, así como compilación en Make.

2. Nota teórica

Las listas enlazadas son conexiones de datos enlazados mediante nodos de manera lineal. Para lograr obtener cualquier elemento de la lista simplemente se requiere el puntero a la primer estructura, ya que este tiene guardado el puntero a la segunda estructura y esto se repite las n veces que sea necesario. La principal ventaja de este tipo de listas con respecto a los arreglos es que no se necesita saber cuál es la cantidad de datos que se va a ingresar ya que utiliza reservas de memoria dinámica, por lo que el usuario podrá ingresar los datos que desee mientras se disponga de memoria. [1]

Vemos en la figura 1 un diagrama de la estructura de una lista enlazada, los cuadros con puntos representan punteros mientras que los cuadros con letras representan datos, notamos que el primer puntero apunta a la primer estructura, que contiene el dato de valor de la lista (lo que se desea guardar) y un puntero hacia la siguiente estructura, de esta forma se tiene toda la estructura junta. Si se quiere agregar un elemento en medio de la lista se debe abrir un espacio para el mismo y hacer que el puntero anterior de donde se va a posicionar ahora apunte al nuevo dato ingresado, mientras que el puntero del dato ingresado debe apuntar al elemento que se va a encontrar después de él. Se debe tener cuidado ya que si se agrega mal el elemento la lista quedará cortada y no funcionará.

En caso de querer eliminar un elemento se debe realizar la operación inversa a la presentada en la figura 1, ya que más bien se deben tomar el puntero anterior y ahora apuntarlo al posterior, mientras que se procede a borrar la memoria del elemento eliminado. Se pueden realizar todas las funciones necesarias para que la lista sea funcional, tales como , ordenar la lista, extraer un dato de la lista, liberar su memoria, etc.

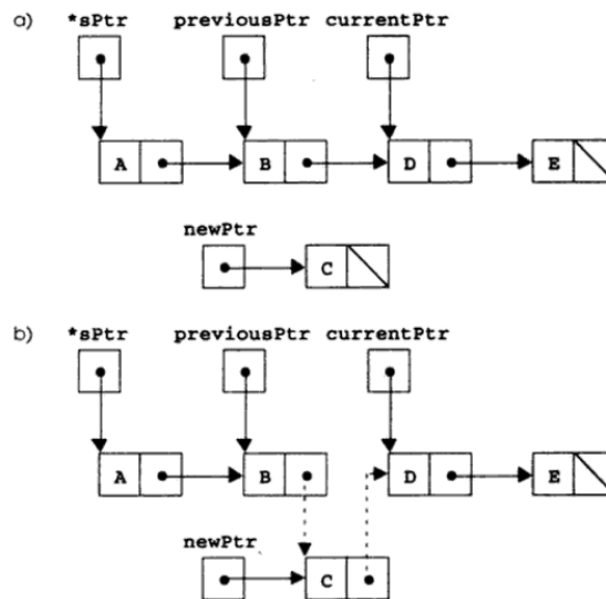


Figura 1: Estructura de una lista enlazada. Tomado de [1]

3. Análisis de resultados

En esta sección se procederá a analizar los resultados obtenidos en la implementación de la biblioteca para listas enlazadas. Se mostrará el funcionamiento de cada una de las funciones con capturas así como su explicación, también el proceso de compilación con make.

3.1. Main y compilación

3.1.1. Compilación utilizando Make

Para la compilación de este proyecto se utilizó un conjunto de directorios organizados como lo vemos en la figura 2 donde en la carpeta `src` se encuentra un archivo `Makefile` junto con los archivos `.c` que contienen el `main` y las funciones programadas, así como un archivo binario que es requerido para una de las funciones y un archivo de texto plano que también se requiere para otra función. En la carpeta `include` se encuentra la inicialización de las funciones en el archivo `.h` que se ve en la imagen, y la carpeta `build` está vacía.

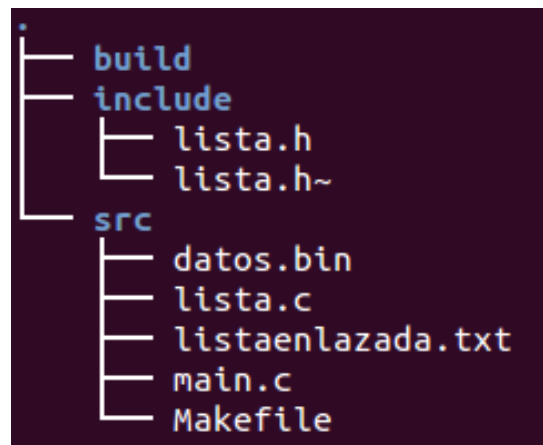


Figura 2: Directorio make antes de compilar con comando make

En la figura 3 vemos que al ejecutar el comando make estando en el directorio src notamos que se compila el programa correctamente, además aparece el archivo main en dicha carpeta que es el que se ejecutará para comenzar el programa. En la carpeta build vemos que quedan los archivos .o para mantener el orden en la carpeta src. A continuación veremos el programa en ejecución.

```

est@yeison-G551JW:~/Escritorio/Laboratorio-7-plataformas-YeisonRP-B56074/Make del proyecto/src$ make
gcc -c -o ../build/main.o main.c -I../include
gcc -c -o ../build/lista.o lista.c -I../include
gcc -o main ../build/main.o ../build/lista.o -I../include
est@yeison-G551JW:~/Escritorio/Laboratorio-7-plataformas-YeisonRP-B56074/Make del proyecto/src$ cd ../
est@yeison-G551JW:~/Escritorio/Laboratorio-7-plataformas-YeisonRP-B56074/Make del proyecto$ tree
.
├── build
│   ├── lista.o
│   └── main.o
├── include
│   ├── lista.h
│   └── lista.h~
└── src
    ├── datos.bin
    ├── lista.c
    ├── listaenlazada.txt
    ├── main
    ├── main.c
    └── Makefile

3 directories, 10 files
est@yeison-G551JW:~/Escritorio/Laboratorio-7-plataformas-YeisonRP-B56074/Make del proyecto$
  
```

Figura 3: Compilando con Make el programa

3.1.2. Ejecutando el programa y explicación del main

Al ejecutar el archivo main generado anteriormente (figura 4) se genera un menú de bienvenida en el cuál el usuario decide si crear su propia lista enlazada desde cero, o cargar una lista enlazada desde el archivo binario del directorio src, que contiene diez mil números enteros. Si el usuario decide crear desde cero la lista se le solicita un número para comenzar con la lista.

```
est@yeison-G551JW:~/Escritorio/Laboratorio-7-plataformas-YeisonRP-B56074/Make del proyecto/src$ ./main
Hola, este programa es una biblioteca para una lista enlazada.
Seleccione la opción que desea con el número indicado en el índice.
1. Crear una lista enlazada desde cero
2. Crear una lista enlazada desde un archivo binario
```

Figura 4: Menú inicial del programa

Cualquiera de las elecciones iniciales del usuario (válidas) lo llevará al menú principal (figura 5) que contiene todas las acciones que se pueden realizar en la lista enlazada.

```
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
```

Figura 5: Menú principal del programa.

Para realizar estos menú se utilizaron if junto con while para controlar la acción que el usuario desea realizar. Todas las funciones que se utilizaron se dejaron intactas del archivo proporcionado para este laboratorio (su inicialización) y en todos los casos se demuestra su correcto funcionamiento usando los parámetros requeridos.

A continuación se explicará y demostrará la funcionalidad de cada uno de los elementos del menú, o sea, se demostrará que las funciones hacen lo que deberían.

3.2. Funciones

3.2.1. `pos_t* createList(int first_value)`

Esta función es la primera que se debe ejecutar en el programa ya que es la encargada de crear el primer puntero de la lista con su primer valor entero. Este puntero es el más importante ya que es el punto de partida para las demás funciones. Dicha función recibe un entero ingresado por el usuario y este será el primer dato de la lista.

Para la creación de esta función se utiliza un puntero de tipo `pos_t` donde `pos_t` es la estructura que contiene dos datos, un `int` que guarda un valor y un puntero tipo `pos_t` que apuntará a la siguiente estructura, en este caso como es el primer dato este apuntará a `NULL`. Es importante resaltar que esta función reserva memoria para un dato tipo `pos_t` que es donde se guardará el primer entero junto con el puntero a la siguiente dirección.

La función retorna el puntero a la primer estructura de la lista, en este programa dicha función se ejecuta en el menú inicial (figura 4) cuando el usuario decide qué hacer, el valor que retorna la función se guarda en una variable `pos_t` tipo puntero dentro del `main`. Esta variable va a ser utilizada en las demás funciones. En la figura 6 vemos que al imprimir la lista después de haber pasado por el

menú inicial solo se encuentra el primer dato ingresado por el usuario, ya que es el que se creó con `createList`.

```
1. Crear una lista enlazada desde cero
2. Crear una lista enlazada desde un archivo binario
1
Ingrese el primer elemento de la lista.
5
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1      Valor: 5
```

Figura 6: Utilizando función `createList`

3.2.2. `void printList(pos_t* head)`

Esta función es la encargada de imprimir la lista cuando el usuario lo solicite, para ejecutarse lo único que necesita es el puntero al primer elemento, la función no tiene ningún retorno.

Para realizar el recorrido por toda la lista se utiliza un `while` y se controla que no se haya llegado al último elemento de la lista verificando si el puntero al siguiente elemento es nulo o no (ya que el último elemento es el único nulo), cuando se llega al último elemento de la lista se sale del `while` e imprime el último dato de la lista y procede a terminar el programa.

Se realizó la función para que muestre también la posición en la que se encuentra el dato mostrado de la lista, ya que otras funciones solicitan dicha posición. En la figura 7 vemos `printList` en funcionamiento.


```
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remove elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1      Valor: 1
Posición: 2      Valor: 5
Posición: 3      Valor: 3
Posición: 4      Valor: 5
Posición: 5      Valor: 2
Posición: 6      Valor: 2
Posición: 7      Valor: 1
Posición: 8      Valor: 6
Posición: 9      Valor: 3
```

Figura 7: Utilizando función printList

3.2.3. int push_back(pos_t* head, int new_value)

Esta función se encarga de crear un elemento al final de la lista, regresa un cero en caso de que el elemento se haya añadido correctamente y un uno en caso contrario. el valor de retorno de esta funcion es analizado por un if para decidir dependiendo si el valor pudo ser o no agregado.

El funcionamiento interno de esta función lo que realiza es una búsqueda por medio de un while de cuál es el último elemento de la lista verificando el momento en el cuál la dirección al siguiente elemento sea nula. Al encontrar el último elemento se crea un puntero auxiliar tipo pos_t en el cuál se reserva memoria y se ingresa el valor de new_value que es el entero que se queria agregar al final de la lista. Ahora debemos apuntar la dirección del último valor (encontrado anteriormente) hacia el nuevo valor final, que sería donde se encuentra el entero new_value.

En la figura 8 vemos la funcion en funcionamiento, al ver la lista vemos que solo hay tres valores, si se elige la posición 2 y luego a agregar el número 0 y volvemos a imprimir la lista, notamos que el cero se agregó al final.

```

La lista actual es:
Posición: 1      Valor: 1
Posición: 2      Valor: 5
Posición: 3      Valor: 3
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
2

Indique el valor del elemento a agregar al final de la lista
0
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1      Valor: 1
Posición: 2      Valor: 5
Posición: 3      Valor: 3
Posición: 4      Valor: 0
¿Qué desea hacer?

```

Figura 8: Utilizando función pushback

3.2.4. `int push_front(pos_t** head, int new_value)`

Esta función se encarga de agregar un elemento al inicio de la lista, para hacerlo recibe dos datos, el entero que se desea agregar y un puntero al puntero del primer elemento, esta función retorna un cero en caso de que se añadiera el elemento correctamente y un uno en caso contrario.

Para lograr el cometido esta función guarda el valor (no la dirección) del puntero doble en un puntero tipo `pos_t`, ya que este valor va a ser la dirección del puntero original. Así que se procede a reservar memoria para el nuevo primer elemento y se guarda la dirección del anterior primer elemento en el puntero `next` del nuevo primer elemento y el valor entero `new_value` se guarda en el nuevo primer elemento. Por último se debe cambiar la dirección del puntero `head` al nuevo puntero inicial.

En la figura 9 vemos la función en funcionamiento, al ver la lista vemos que solo hay cuatro valores, si se elige la posición 3 del menú y luego a agregar el número 0 y volvemos a imprimir la lista, notamos que el cero se agregó al inicio.

```

La lista actual es:
Posición: 1      Valor: 1
Posición: 2      Valor: 5
Posición: 3      Valor: 3
Posición: 4      Valor: 0
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remove elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
3

Indique el valor del elemento a agregar al inicio de la lista
0
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remove elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1      Valor: 0
Posición: 2      Valor: 1
Posición: 3      Valor: 5
Posición: 4      Valor: 3
Posición: 5      Valor: 0

```

Figura 9: Utilizando función pushfront

3.2.5. `int pop_front(pos_t **head)`

La función `pop_front` elimina el elemento inicial de la lista y retorna el valor del elemento que se eliminó, la función recibe solamente un puntero al puntero donde se encuentra el primer elemento.

Para eliminar el primer elemento la función guarda en una variable auxiliar la dirección a la segunda posición de la lista y en otra variable el valor entero de la primera posición de la lista (para retornarlo), se procede a liberar la memoria del antiguo primer elemento y ahora se hace que el puntero `head` (puntero inicial) sea igual al segundo elemento de la lista anterior, por lo que este sería el nuevo primer elemento.

Vemos en la figura 10 que si elegimos la opción de eliminar un elemento al comienzo de la lista nos retorna el valor eliminado, además notamos que la imprimir nuevamente la lista el valor eliminado ya no se encuentra.

```

La lista actual es:
Posición: 1      Valor: 9
Posición: 2      Valor: 7
Posición: 3      Valor: 5
Posición: 4      Valor: 6
Posición: 5      Valor: 3
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remove elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
5

El número eliminado al comienzo es 9

¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remove elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1      Valor: 7
Posición: 2      Valor: 5
Posición: 3      Valor: 6
Posición: 4      Valor: 3
¿Qué desea hacer?

```

Figura 10: Utilizando función pop_front

3.2.6. int pop_back(pos_t* head)

Esta función es similar a la anterior pero ahora elimina el elemento final de la lista, también retorna el elemento eliminado pero en lugar de recibir un puntero doble recibe un puntero simple.

Para eliminar el elemento final primeramente se debe utilizar un while que termine cuando el puntero next sea igual a nulo, en ese momento tendremos el elemento final. También es importante ir guardando en una variable auxiliar el elemento anterior, para que cuando se encuentre el elemento final se tenga el último y el antepenúltimo. Ahora se procede a guardar la variable entera del elemento final para retornarla y se libera la memoria de dicha estructura, ahora al elemento antepenúltimo se le indica que la dirección next a la que apunta sea nula para que se convierta en el elemento final.

Vemos en la figura 11 que si elegimos la opción de eliminar un elemento al final de la lista nos retorna el valor eliminado, además notamos que la imprimir nuevamente la lista el valor eliminado ya no se encuentra.

```

La lista actual es:
Posición: 1      Valor: 7
Posición: 2      Valor: 5
Posición: 3      Valor: 6
Posición: 4      Valor: 3
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
4

El número eliminado al final es 3

¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1      Valor: 7
Posición: 2      Valor: 5
Posición: 3      Valor: 6

```

Figura 11: Utilizando función pop_back

3.2.7. int insertElement(pos_t** head, int pos, int new_value)

Esta función ingresa a la lista (en cualquier posición) un elemento nuevo que se pide como parámetro, también se pide la posición donde se quiere ingresar dicho entero. La función regresa un cero si fue posible agregar el número, si hubo problemas con la reserva de memoria retorna un 1.

Para lograr el cometido esta función recorre el arreglo hasta llegar a la posición que el usuario ingresó (si la posición no está disponible la función retorna un 1 y no realiza ningún cambio), para recorrer el arreglo se utiliza un while cuyo final es cuando la dirección al siguiente elemento next sea nula. Mientras se recorre todo el arreglo un if verifica si ya se llegó a la posición correcta, si es así entonces se reserva memoria para un nuevo elemento, a este nuevo elemento se le agrega el valor de new_value en data y su puntero next apunta hacia el elemento que se encontraba en la posición indicada por el usuario, el puntero next que se encontraba un lugar antes de la posición indicada por el usuario ahora apunta al nuevo elemento ingresado. Esta función en resumen abre un campo entre dos valores y se ingresa en medio, si el usuario eligiera el valor final del arreglo se llama una de las funciones hechas anteriormente que realizan esto, igual si el usuario decide ingresar un número en la primera posición del arreglo.

En la figura 12 vemos que si se elige la opción 6 que corresponde a ingresar un elemento, y se dice que el elemento a ingresar tenga un valor de 0 y sea en la posición 2 al imprimir la lista notamos que

ahora el elemento 0 ocupa esta posición, y el que estaba en esta posición se movió una hacia arriba.

```

La lista actual es:
Posición: 1      Valor: 7
Posición: 2      Valor: 5
Posición: 3      Valor: 6
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
6

Indique el valor del elemento a agregar en la lista
0
Indique la posición del elemento a agregar en la lista
2
El elemento fue agregado a la lista
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1      Valor: 7
Posición: 2      Valor: 0
Posición: 3      Valor: 5
Posición: 4      Valor: 6
¿Qué desea hacer?

```

Figura 12: Utilizando función insert element

3.2.8. int removeElement(pos.t** head, int pos)

Esta función elimina un elemento que se encuentre en la posición pos ingresada por el usuario utilizando como parámetro de entrada un puntero al puntero que contiene el valor inicial.

Primero se va a explicar el caso especial en el que el usuario quiera eliminar el primer elemento de la lista, si esto ocurre esta función invoca a la función pop front y esta se encarga de realizar esta tarea, si el número no se encuentra en el primer lugar entonces se debe revisar toda la cadena de números.

En esta acción que se quiere realizar se debe tener información de tres punteros a estructuras, primeramente la estructura que se desea eliminar, una posición antes y una después, para esto se utilizan tres variables auxiliares. Se procede a recorrer todo el arreglo con un while que termina cuando la variable auxiliar que se encuentra en medio llega a que su valor next es nulo, si en algún momento el entero pos ingresado por el usuario coincide con un contador (para saber en cuál posición se está) entonces se procede a realizar la acción de eliminar el elemento.

Para eliminar el elemento simplemente hacemos un free al mismo, pero debemos cambiar que el elemento que estaba antes ahora apunte al elemento que estaba después, con esto ya estaría eliminado el número y la lista seguiría funcionando.

Si se recorre todo el while y se llega al final del arreglo y el contador es igual a pos entonces significa que el usuario quiere eliminar la última posición, así que se llama a la función pop back para que se encargue de esto. Al eliminar un número la función retorna el número que eliminó.

En la figura 13 vemos que al elegir la opción de eliminar un elemento si se selecciona la posición 3, se retorna que se eliminó el número 5, y al volver a imprimir la lista este valor ya no se encuentra.

```
La lista actual es:
Posición: 1 Valor: 7
Posición: 2 Valor: 0
Posición: 3 Valor: 5
Posición: 4 Valor: 6
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
7
Indique la posición del número en la lista que desea eliminar. comenzando por 1
3
Se eliminó el numero 5

¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1
La lista actual es:
Posición: 1 Valor: 7
Posición: 2 Valor: 0
Posición: 3 Valor: 6
¿Qué desea hacer?
```

Figura 13: Utilizando función remove element

3.2.9. int getElement(pos_t* head, int index, int* valid)

Esta función se encarga de retornar el elemento que se encuentra en una posición indicada por el usuario, cuenta con un puntero a un entero que se pone en 1 si el dato retornado no es válido.

Para encontrar el número a retornar se utiliza un while que se detenga hasta que se encuentre el último elemento del arreglo, se tiene un contador que sirve para compararlo con index y saber si se llegó a la posición indicada, esto se realiza con un if y cuando se llega a dicha posición se guarda el valor data en una variable que se va a retornar, se utiliza la función goto para salir del while en este caso.

Vemos en la figura 13 que si queremos obtener un elemento de la lista ingresamos su posición, la función nos retornará el mismo sin cambiar nada en la lista acutal.

```

La lista actual es:
Posición: 1      Valor: 7
Posición: 2      Valor: 0
Posición: 3      Valor: 6
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
8

Cual es la posición del número en la lista que desea encontrar
2
El número encontrado tiene el valor de 0

```

Figura 14: Utilizando función get element

3.2.10. void sort(pos_t* head, char dir)

Esta función se encarga de ordenar la lista de manera ascendente si recibe un char a y de manera descendente si recibe un char d. Se cuenta también con el puntero al inicio del arreglo.

Para realizar este ordenamineto primer se verifica con un if si el char ingresado coincide con a o con d, para decidir así qué método usar. Al verificar esto se procede a hacer ordenamiento por el método burbuja, para esto requerimos dos punteros a estructuras, uno que vaya adelante del otro para compararlos, se recorre toda la estructura con un while para saber cuando se llega al último elemento. Los elementos se van comparando uno a uno, si uno es mayor que el otro (dependiendo de si es ascendente o descendente) se intercambian su valor data y una variable controladora se pone en cero. Todo este while se encuentra dentro de otro que en el comienzo reinicia los valores de los punteros tipo pos_t y coloca la variable controladora en 1, si se recorre todo el arreglo y dicha variable sigue en 1 entonces el while principal termina y esto quiere decir que ya se terminó de ordenar los elementos.

Vemos en la figura 15 que al solicitar ordenar el arreglo de manera ascendente (Se solicita con un número en el menú, pero en el código lo que se ingresa es un char con la letra a) e imprimirlo notamos que está ordenado de menor a mayor.


```
La lista actual es:
Posición: 1 Valor: 7
Posición: 2 Valor: 0
Posición: 3 Valor: 6
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
9

Ingrese un 0 para ordenar la lista de manera ascendente o un 1 de manera descendente
0
La lista fue ordenada de manera ascendente

¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1

La lista actual es:
Posición: 1 Valor: 0
Posición: 2 Valor: 6
Posición: 3 Valor: 7
```

Figura 15: Utilizando función sort de manera ascendente

En la figura 16 que al solicitar ordenar el arreglo de manera descendente (Se solicita con un número en el menú, pero en el código lo que se ingresa es un char con la letra d) e imprimirlo notamos que está ordenado de mayor a menor.

```

La lista actual es:
Posición: 1 Valor: 0
Posición: 2 Valor: 6
Posición: 3 Valor: 7
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
9
Ingrese un 0 para ordenar la lista de manera ascendente o un 1 de manera descendente
1
La lista fue ordenada de manera descendente

¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto.
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
1
La lista actual es:
Posición: 1 Valor: 7
Posición: 2 Valor: 6
Posición: 3 Valor: 0

```

Figura 16: Utilizando función sort de manera descendente

3.2.11. `pos_t* readList(const char* filePath)`

Esta función recibe un `char` con la dirección del archivo binario que se desea leer para extraer los números enteros y regresa un puntero al primer elemento del arreglo creado con estos datos.

Para lograr hacer una lista con los datos se utiliza un `for` que va de 1 a 10000 (Ya que estas es la cantidad de datos del archivo binario) que recorra todos los datos enteros para poder extraerlos, pero se debe tener cuidado ya que el primer dato que se extrae de la lista se debe ingresar en la función `createList` y guardar este puntero en una variable auxiliar, esto con el fin de tener el inicio de la lista, los siguientes 9999 datos se extraen y se colocan en la función `push back` para que se vayan ingresando.

Si en el menú inicial de la figura 4 se elige la opción 2 se invocará la función `readList`, vemos en la figura 17 que al imprimir la lista vemos que se cargaron los enteros que estaban dentro del archivo binario.

```
Posición: 9980      Valor: 150
Posición: 9981      Valor: 209
Posición: 9982      Valor: 164
Posición: 9983      Valor: 13
Posición: 9984      Valor: 22
Posición: 9985      Valor: 207
Posición: 9986      Valor: 184
Posición: 9987      Valor: 245
Posición: 9988      Valor: 202
Posición: 9989      Valor: 37
Posición: 9990      Valor: 212
Posición: 9991      Valor: 78
Posición: 9992      Valor: 190
Posición: 9993      Valor: 137
Posición: 9994      Valor: 98
Posición: 9995      Valor: 13
Posición: 9996      Valor: 32
Posición: 9997      Valor: 116
Posición: 9998      Valor: 67
Posición: 9999      Valor: 172
Posición: 10000     Valor: 220
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
```

Figura 17: Utilizando función read list

3.2.12. void writeList(pos_t* head, const char* filePath)

La función writeList recibe como parámetros de entrada el puntero al primer elemento de la lista y un char que contiene la dirección del archivo de texto que se quiere utilizar para escribir los elementos de la lista.

Se utiliza un while para recorrer todos los datos de la lista enlazada, el while se detiene cuando se llega a que el siguiente valor del puntero es nulo. Mientras está en el loop va imprimiendo en el archivo de texto cada dato entero que encuentra en la lista.

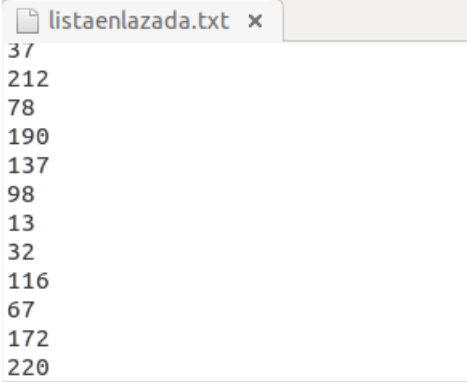
En la figura 18 vemos que se guardaron los archivos que fueron mostrados anteriormente (los del archivo binario) en el archivo, en la figura 19 vemos que al revisar el archivo los elementos finales coinciden con los de la lista.

```

Posición: 9990      Valor: 212
Posición: 9991      Valor: 78
Posición: 9992      Valor: 190
Posición: 9993      Valor: 137
Posición: 9994      Valor: 98
Posición: 9995      Valor: 13
Posición: 9996      Valor: 32
Posición: 9997      Valor: 116
Posición: 9998      Valor: 67
Posición: 9999      Valor: 172
Posición: 10000     Valor: 220
¿Qué desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
10
Se guardó la lista en el archivo de texto.

```

Figura 18: Utilizando función writelist en el menú



```

listaenlazada.txt x
37
212
78
190
137
98
13
32
116
67
172
220

```

Figura 19: Verificando función writelist

3.2.13. int freeList(pos_t* head)

Esta función se encarga de liberar la memoria reservada en la lista enlazada, para esto solo recibe el puntero al primer elemento.

Para limpiar toda la lista se utiliza una función recursiva que entre en sí misma las veces que sea necesario hasta que se llegue a que el valor next es nulo, cuando esto ocurre libera la memoria de donde se encuentra y va en retroceso limpiando la memoria de todo el arreglo hasta llegar al primer elemento.

En la figura 20 vemos que al ingresar el número 11 esta función libera toda la memoria almacenada en la lista y termina el programa.

```
¿Que desea hacer?
1. Imprimir la lista.
2. Agregar elemento al final de la lista.
3. Agregar elemento al inicio de la lista.
4. Eliminar elemento al final de la lista.
5. Eliminar elemento al inicio de la lista.
6. Insertar elemento a la lista.
7. Remover elemento de la lista.
8. Obtener elemento de la lista.
9. Ordenar la lista.
10. Guardar lista en archivo de texto
11. Borrar memoria de la lista. (el programa terminará)
12. Salir del programa (se borrará la memoria de la lista)
11
est@veison-G551JW:~/Escritorio/Laboratorio-7-plataformas-VeisonRP-B56074/Make del proyecto/src$
```

Figura 20: Verificando función freelist

4. Conclusiones y recomendaciones

Como se pudo observar en este informe es necesario tener un dominio de todos los temas vistos en el curso con relación a C para poder programar todas las funciones necesarias para la biblioteca de la lista enlazada, también se nota la utilidad de las listas enlazadas a la hora de manejar datos ya que se pueden agregar los que sean necesarios porque la misma lista crea memoria para almacenarlos.

Se comprobó la utilidad de la compilación utilizando Make cuando se tiene gran cantidad de funciones, así como el potencial que representan los punteros a la hora de trabajar con datos. También se vió que la implementación de lectura y escritura de archivos de texto y binarios puede facilitar los procesos de transmisión de datos al programa.

Se recomienda tener especial cuidado a la hora de liberar la memoria de la lista ya que si no se hace bien podría traer problemas a largo plazo. También se debe tener precaución a la hora de hacer cualquier modificación en la lista ya que se si daña algún puntero de la lista ya no será funcional.

Referencias

- [1] Deitel, P. Deitel, H. (1995). *Cómo programar en C/C++*. Pearson. 471.