Laboratorio 7
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
Programación bajo Plataformas Abiertas
IE0117

Luis Daniel Ferreto Chavarría
Carné: B82958
Profesor: MSc. Marco Villalta Fallas

24 de febrero de 2021

0.1. Introducción

El presente laboratorio, consiste en la elaboración de una librería que contiene métodos que pueden ser aplicados a una lista enlazadas, así como la estructura que la define, entre las cuales destacan, la creación de la lista enlazada, insertar elementos según la posición que se indique en la lista, remover elementos de la lista e imprimirla. Todos estos métodos pueden ser llamados en la función principal y con la creación de variables dentro de este entorno, es posible llamar estos métodos y mostrarlo en la terminal (LInux) o consola (Windows).
La función principal ,en la que se incluye el archivo de encabezado para la ejecución del programa.
Para la implentación de necesita de reserva de memoria dinámica, uso de punteros, manejo de archivos, estructuras y listas enlazadas. Además el programa se compila de manera automatizada con un makefile, donde el arbol de directorios utilizados se muestra en 10 y dentro del directorio esta el Makefile con las instrucciones necesarias para compilar el programa con los requerimientos del laboratorio, además mostrar información necesaria para depurar errores.
Dado que los archivos se encuentran en directorios separados, para mostrar los resultados y el código fuente completo, se compactaron los métodos de la librería en el archivo principal y en este por medio de diagramas se expicaron los resultados obtenidos en terminal, donde cada método del programa afue documentado en doxygen, donde se adjunta el doxyfile correspondiente.
Se utilizó el comando size en linux para observar la segmentación de memoria del programa, así como valgrind para comprobar que el programa no posee "*memory leaks*"ni errores.

0.1.1. Nota Teórica

Según [1], la lista es una estructura dinámica de datos, por este hecho para crear un nodo en una lista, se debe hacer la reserva respectiva de memoria dinámica. cada nodo está formado por los datos junto con el puntero al siguiente objeto, donde los punteros no necesariamente tienen que estar en posiciones consecutivas de memoria.
En una lista existe un elemento que es la raíz o cabeza que empieza la lista al cual se le almacena un dato y el nodo apunta a un siguiente nodo y así consecutivamente hasta que el último elemento, apunta al último elemento en la lista.
Cada dato del nodo de la lista, es una estructura de datos en el lenguaje C, donde el sentido de creación de la lista puede ser tanto de derecha a izquierda o de derecha izquierda, según explica **??**, para efectos del desarrollo de este laboratorio, la lista se desarrolló de izquierda a derecha.
Por último, cabe destacar que las listas también pueden ser doblemente enlazadas, lo cual implica que cada nodo apunta a dos direcciones con dos punteros, donde uno apunta ala elemento previo y el otro al elemento siguiente, con lo cual se puede recorrer la lista en dos sentidos distintos.
Según [1], las listas se definen en C como:

```
struct lista { /* lista simple enlazada */
  struct dato datos;
  struct lista *sig;
};
```

Para inicializar la lista se hace con:

```
struct lista *l; /* declaraci n */
l = NULL; /* inicializaci n */
```

En este laboratorio, al declaración y la asignación, se hizo en una misma línea.
Según [2] para verificar las fugas de memoria en un programa, en el sistema operativo Linux, es posible usar el software Valgrind para detectar esta clase de errores y además depurar el programa, en otras palabras funciona como depurador de memoria, donde si existen errores, este no indica la línea en que sucedió, por lo tanto para ver la información de depuración de memoria con la línea respectiva, se tiene que compilar el programa como $\boxed{gcc -l -Wall -o\ main\ main.c}$ y para el valgrind $\boxed{valgrind -l --leak-check=yes ./main}$ y de esta manera se pueden observar las fugas de memoria y la línea donde sucedió el error. Según [2], esta clase de errores se pueden dar al no liberar memoria dinámica que fue reservada en el programa. Una buena práctica luego de liberar la memoria es igualar el puntero al que se le reservó la memoria a NULL para evitar que siga apuntando a una dirección ya ocupada en la memoria RAM. Además de esto es importante siempre inicializar los punteros para que estos no apunten a direcciones aleatorias en el inicio del programa y no sucedan fugas de memoria en la compilación del programa.
Según [3], los archivos son datos que se almacenan en un dispositivo de memoria, los cuales contienen datos estructurados en colecciones ya sean de entidades elementales o básicas. Los archivos pueden ser de texto o binarios, donde los binarios son una secuencia de bytes, los cuales poseen una relación de uno a uno con el dispositivo de memoria y no tiene

lugar a traducciones de caracteres, sino que pasa ser leídos, se necesita de un programa que los pueda leer desde el dispositivo de memoria, donde los datos leídos, son los mimos que los escritos en el archivo. En el lenguaje de programación C, se hace una asociación de una secuencia con un archivo específico realizando la operación de apertura del archivo y con esto, la información dentro del archivo puede ser intercambiada con el programa. Algunas operaciones que se pueden hacer para el manejo de archivos en C, son:

- fopen()
- fclose()
- fgets()
- fputs()
- fprintf()
- fscanf()
- feof()
- fflush()

Para el desarrollo de las listas enlazadas, se usaron esturas en el lenguaje C, por lo que a continuación se documentan:
Según [4] las estructuras son colecciones de variables que están relacionadas bajo un etiqueta. Este tipo de dato, pueden contener variables de diferente tipo de dato, lo contrario de los arreglos que solo pueden contener datos de un tipo de variable.
Las estructuras antes de usarse deben ser declaradas, pueden ser en el programa principal o en un archivo de encabezados. Para inicializarse se puede con el operador punto según la variable correspondiente o mediante corchetes, un ejemplo [4]:

```
struct fecha f = f4, 7, 1776, 186, "Julio"g;
```

Además las estructuras se pueden usar con funciones, se puede usar un typedef para facilitar el uso de etiquetas [4].

0.2. Desarrollo

Para simplicidad de mostrar y explicar el desarrollo, se compactaron las funciones que contiene la librería lista.h en el archivo de código fuente main.c y se adjunta el código implementado para el mismo. Este código contiene las macros necesarias para ejecutar el programa y las funciones aparecen declaradas en estilo general y se llaman en la función principal.
En esta sección se dará por ende, énfasis a explicar el comportamiento general de las funciones implementadas para las listas enlazadas y de la estructura de la lista en la enlazada, ya que las funciones de la librería han sido debidamente documentadas en doxygen.

0.2.1. Código fuente del programa

```
#include <stdio.h>
#include <stdlib.h>
#define NOMBRE "listaEnlazada.bin"
typedef struct node {
    int data; //**< data */
    struct node* next; //**< pointer to next element */
} node;

int push_back(node* head, int new_value)
{
    node* x = NULL;
    node* nuevo = NULL;

    x = head;
```

```
    while (x->next != NULL)
    {
        x = x->next;
    }

    nuevo = (node*) malloc(sizeof(node));

    if (nuevo == NULL)
    {
        return 1;
    }

    nuevo->data = new_value;
    nuevo->next = NULL;
    x->next = nuevo;
    return 0;
}
int push_front(node** head, int new_value)
{
    node* nuevo = (node*) malloc(sizeof(node));
    if (nuevo == NULL)
    {
        return 1;
    }
    nuevo->data = new_value;
    nuevo->next = *head;
    *head = nuevo;
    return 0;
}
void printElement(int value)
{
    printf("%d ", value);
}
node* readList(const char* filePath)
{
    int data;
    node *cabeza = NULL;
    node *x = NULL;
    node *y = NULL;
    FILE *F = fopen(filePath, "rb");

    if (F != NULL)
    {
        cabeza = (node*) malloc(sizeof(node));
        fread(&cabeza->data, sizeof(int), 1, F);
        cabeza->next = NULL;
        y = cabeza;
        printf("dato leido: %d\n", cabeza->data);

        while (!feof(F))
        {
            fread(&data, sizeof(int), 1, F);
            if (!feof(F))
            {
                printf("dato leido: %d\n", data);
                x = (node*) malloc(sizeof(node));
```

```
                x->data = data;
                x->next = NULL;
                y->next = x;
                y = x;
            }
        }

        fclose(F);
    }
    return cabeza;
}
void writeList(node* head, const char* filePath)
{
    node* x = NULL;
    FILE *F = fopen(filePath, "wb");
    if (F != NULL)
    {
        x = head;
        while (x != NULL)
        {
            fwrite(&x->data, sizeof(int), 1, F);
            printf("dato escrito: %d\n", x->data);

            x = x->next;
        }

        fflush(F);
        fclose(F);
    }
}
node* createList(int first_value)
{
    node *nuevo = NULL;

    nuevo = (node*)malloc(sizeof(node));
    if( nuevo != NULL)
    {
        nuevo->data = first_value;
        nuevo->next = NULL;
    }
    return nuevo;
}
void insertElement(node** phead, int pos, int new_value)
{
    node *reco = *phead;
    int cant = 0;
    while (reco != NULL)
    {
        cant++;
        reco = reco->next;
    }
    if (pos <= cant + 1)
    {
        node *nuevo=NULL;
        nuevo=createList(new_value);
        if (pos == 1)
```

```
        {
            nuevo->next = *phead;
            *phead = nuevo;
        }
        else
        {
            if (pos == cant+ 1)
            {
                node *reco = *phead;
                while (reco->next != NULL)
                {
                    reco = reco->next;
                }
                reco->next = nuevo;
                nuevo->next = NULL;
            }
            else
            {
                node *reco = *phead;
                int f;
                for (f = 1; f <= pos - 2; f++)
                {
                    reco = reco->next;
                }
                node *siguiente = reco->next;
                reco->next = nuevo;
                nuevo->next = siguiente;
            }
        }
    }
}
void printList(node* head)
{
    int contador = 0;
    node *n = head;
    while( n != NULL)
    {
        printElement(n->data);
        ++contador;
        printf("contador: %d\n", contador);
        n = n->next;
    }
    printf("\n");
}
void freeList(node* head)
{
    node *reco = head;
    node *bor;
    while (reco != NULL)
    {
        bor = reco;
        reco = reco->next;
        free(bor);
    }
}
void sort(node* head, char dir)
```

```
{
    node *f=head;
    //struct nodo *f1=raiz;
    //f1=f1->sig;
    int *x;
    node *reco = head;
    int cant = 0;
    while (reco != NULL)
    {
        cant++;
        reco = reco->next;
    }
    x=(int *)malloc(cant*sizeof(int));
    for(int i=0;i<cant;i++)
    {
        x[i]=f->data;
        f=f->next;
    }
    if(dir =='a')
    {
        int aux;
        //Ordenamiento por metodo burbuja con punteros
        for(int i=0;i<cant;i++)
        {
            for(int j=0;j<cant-1;j++)
            {
                if(*(x+j+1) > *(x+j))
                {
                    aux = *(x+j);
                    *(x+j) = *(x+j+1);
                    *(x+j+1) = aux;

                }
            }
        }
    }
    else if(dir=='b')
    {
        int aux;
        //Ordenamiento por metodo burbuja con punteros
        for(int i=0;i<cant;i++)
        {
            for(int j=0;j<cant-1;j++)
            {
                if(*(x+j+1) < *(x+j))
                {
                    aux = *(x+j);
                    *(x+j) = *(x+j+1);
                    *(x+j+1) = aux;

                }
            }
        }
    }
    for(int i=0;i<cant;i++)
    {
```

```
        printf("%i ",x[i]);

    }
    free(x);
    x=NULL;
    printf("\n");
}
int pop_front(node **phead)
{
    int informacion = (*phead)->data;
//    node *bor = *phead;
    node *aux=(*phead)->next;
    free(*phead);
    *phead = aux;
    return informacion;
}


int pop_back(node* head)
{
    node* x = NULL;
    int cant = 0;
    int data;

    if (head != NULL)
    {
        x = head;
        while (x != NULL)
        {
            cant++;
            x = x->next;
        }
        if (cant == 1)
        {
            data = head->data;
            free(head);
            return data;
        }
        else if (cant > 1)
        {
            x = head;
            while (x->next->next != NULL)
            {
                x = x->next;
            }
            data = x->next->data;
            free(x->next);
            x->next = NULL;
            return data;
        }
    }
    return -1;
}

int removeElement(node** phead, int pos)
{
```

```
    node *reco = *phead;
    int cant = 0;
    while (reco != NULL)
    {
        cant++;
        reco = reco->next;
    }
    if (pos <= cant)
    {
        int informacion;
        node *bor;
        if (pos == 1)
        {
            return pop_front(phead);
        }
        else
        {
            node *reco;
            reco = *phead;
            int f;
            for (f = 1; f <= pos - 2; f++)
            {
                reco = reco->next;
            }
            node *prox = reco->next;
            reco->next = prox->next;
            informacion = prox->data;
            bor = prox;
            free(bor);
            return informacion;

        }

    }
    else
    {
        return -1;
    }
}
int getElement(node* head, int index, int* valid)
{
    node *nAux = head;
    int y;
    for(int i =0;i<index;i++)
    {
        y=nAux->data;
        nAux=nAux->next;
    }
    while( nAux != NULL )
    {
        if(nAux->data == y)
            *valid = 0;
        nAux = nAux->next;
    }
    *valid= 1;
```

```
    return y;
}
int main(int argc, char **argv, char const **envp)
{
    (void) argc;
    (void) argv;
    (void) envp;

    int elemento=1;
    int valid;
    node *cabeza = NULL;
    insertElement(&cabeza, 1, 10);
    insertElement(&cabeza, 2, 7);
    insertElement(&cabeza,3,9);

    writeList(cabeza, NOMBRE);
    node *nuevaLista = readList(NOMBRE);
    printf("lista leida del archivo\n");
    printList(nuevaLista);

    printf("lista creada con codigo en main\n");
    push_back(cabeza, 100);
    push_front(&cabeza, 500);
    printList(cabeza);

    printf("\nEl elemento en la posicion %i es %i\n",elemento,getElement(cabeza, elemento, &valid));
    if(valid==0)
    {
        printf("Elemento no valido\n");
    }
    else if(valid==1)
    {
        printf("Elemento valido\n");
    }

    printf("\nLista luego de eliminar elemento\n");
    int a=removeElement(&cabeza, 3);
    printList(cabeza);
    printf("\nEl elemento eliminado fue: %i\n",a);
    printf("Lista luego de eliminar el ultimo elemento\n");
    int b=pop_back(cabeza);
    if(b!=-1)
    {
        printList(cabeza);
        printf("El ultimo elemento en eliminarse fue %i\n",b);
    }
    else
    {
        printf("La lista esta vacia\n");
    }
    printf("La lista ordenada de manera descendente es: \n");
    //fflush(stdout);
    sort(cabeza, 'a');
    printf("\nLa lista ordenada de manera ascendente es: \n");
    sort(cabeza, 'b');
    freeList(cabeza);
```

```
    return 0;
}
```

0.2.2. Resultado en consola

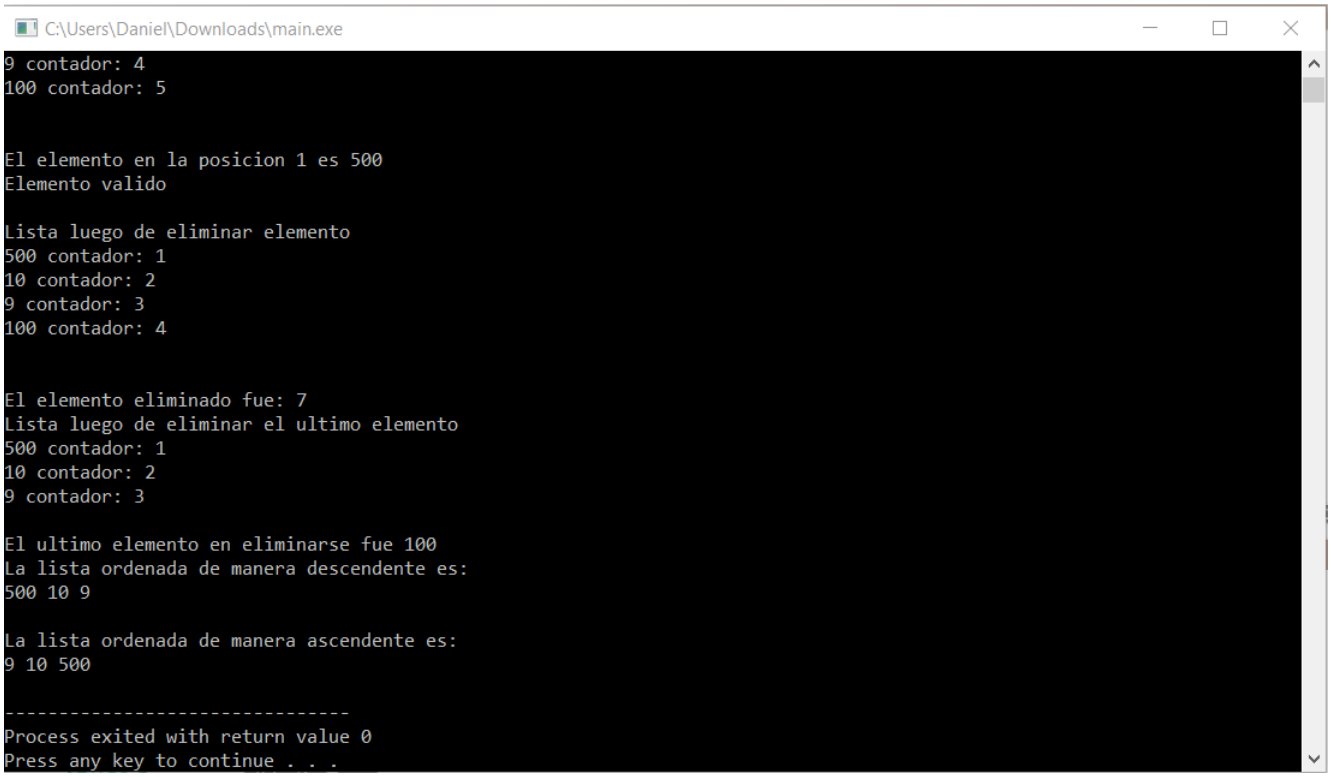

Figura 1: Resultado en consola del programa completo en la función principal

0.2.3. Explicación del procedimiento de creación de la lista con diagramas

Lo primero que se hace en el programa principal, es crear un puntero que apunte a la estructura node y que esta tenga un valor igual al macro NULL, luego de esto, se insertó el primer elemento a la lista enlazada, el cual corresponde al 10. Insertar este elemento implicó la creación de la lista ya que cabeza = NULL y por ende, el programa entiende que la operación no es crear una nueva cabeza para la lista, sino crearla y aparte de esto, que sea la creación de la lista. La función para insertar elemento, recibe la dirección del puntero node, el elemento y la posición del elemento que se desea ingresar, el proceso gráfico se muestra en el la figura 2.
Para definir la creación de la lista, se cambia data que estaba en NULL por el número entero que se desea ingresar a la lista yq eu el siguiente elemento sea NULL para definir que sea el fin de la lista.

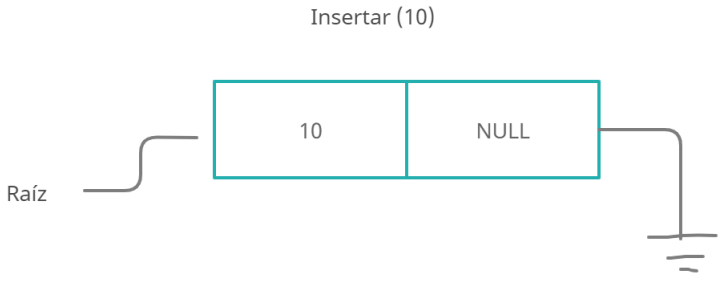

Figura 2: Inserción del primer elemento en la lista (raíz)

Luego de haber creado la lista insertando el primer elemento se insertaron dos elementos más con la función insertElement, los cuales fueron el 7 en la posición 2 y el 9 en la posición 3 y lo que se hizo fue sustituir el siguiente de la cabeza de NULL al dato 7 y el siguiente del nodo con 7 contenido por NULL, seguidamente, cambiar el next del nodo 7 de NULL al dato 9 y el siguiente de 9 por NULL para que este sea el final de la lista. Este proceso se continúa para agregar más elemento a la lista.
Cabe destacar que en este caso se agregaron de manera contigua, es decir, uno después de otro, sin embargo, si la posición al insertar un elemento coincide con la de una elemento ya creado, lo que hace la función es insertar el elemento en la posición ya creada y que el siguiente sea el elemento que tenía esa posición anteriormente.

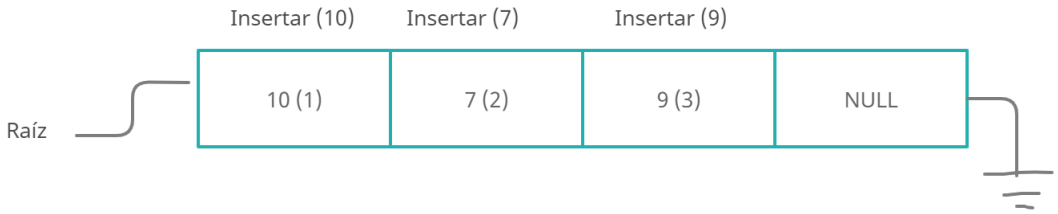

Figura 3: Inserción de los demás elementos luego de haber insertado la raíz

Luego de crear la lista e insertar elementos en la función principal con el método insertList, se utilizaron las funciones push_front y push _back para insertar un elemento al principio de la lista y al final de lista respectivamente. Primeramente se implementó push_front para insertar un nuevo elemento al inicio, el cual corresponde a 500. Para insertarlo hay que tener el cuidado de que el primer elemento corresponde a la cabeza de la lista y por ende, la lista luego de cambiar su cabeza, debe seguir siendo válida. Se indicó que el nodo nuevo que ocuparía la cabeza, tuviera como dato el 500 y como siguiente el segundo elemento, el cual es el mismo siguiente de la cabeza actual y luego de hacer esto, se hizo que el elemento de la raíz ahora no fuera el 10, sino el 500 y el siguiente el 10 y como el 10 tenía como siguiente los demás elemento de la lista, no se perdió el carácter y funcionalidad de la lista.
La inserción del último elemento con push_back tiene una lógica relativamente más sencilla que la inserción del primer elemento, ya que por definición de la lista al crearla, el último elemento, corresponde a NULL, por lo que se recorrió toda la lista y el elemento que tuviera como siguiente el NULL, se cambió su siguiete por el valor del elemento que se deseaba insertar y el siguiente del nodo elemento, se cambió por NULL y así se tuvo un nuevo final de la lista. Esto se muestra gráficamente en la figure 4.

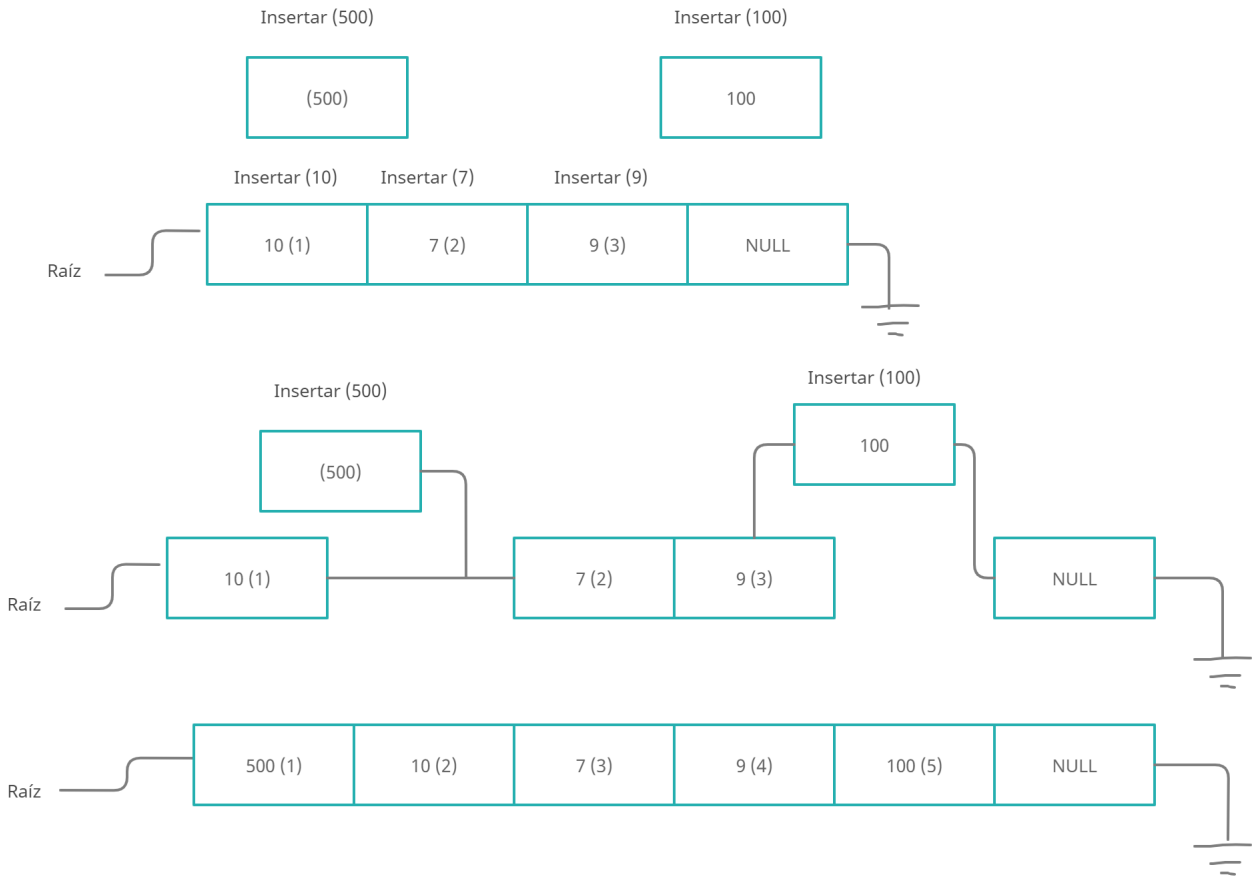

Figura 4: Inserción de nueva raíz y último elemento de la lista

La creación de cada nodo en la lista, se hace por medio de la reserva dinámica, en este caso, se realizó por medio de la función malloc, por lo que para la eliminación de nodos en la lista, se utilizó la función free.
En la función principal se llamó a la función removeElement para eliminar un elemento de la lista, donde los parámetros que recibe esta, es la dirección del puntero cabeza y la posición del elemento que desea eliminar. En este caso se indicó que fuera la 3 y se eliminó el 7 y se mostró en consola la lista sin este elemento y se indicó cual elemento se eliminó, dado que esta función retorna el entero del elemento eliminado.
Para eliminar el tercer elemento, se indicó que el elemento anterior al 7, tuviera como siguiente, el mismo siguiente del 7 y ya no el 7 y una vez hecho esto, se indicó que se liberara la memoria del nodo 7 y no habría problema en la lista, ya que el 10 continuaba la lista con normalidad y no era necesario el 7 en la lista. Esto se muestra gráficamente en la imagen 5.

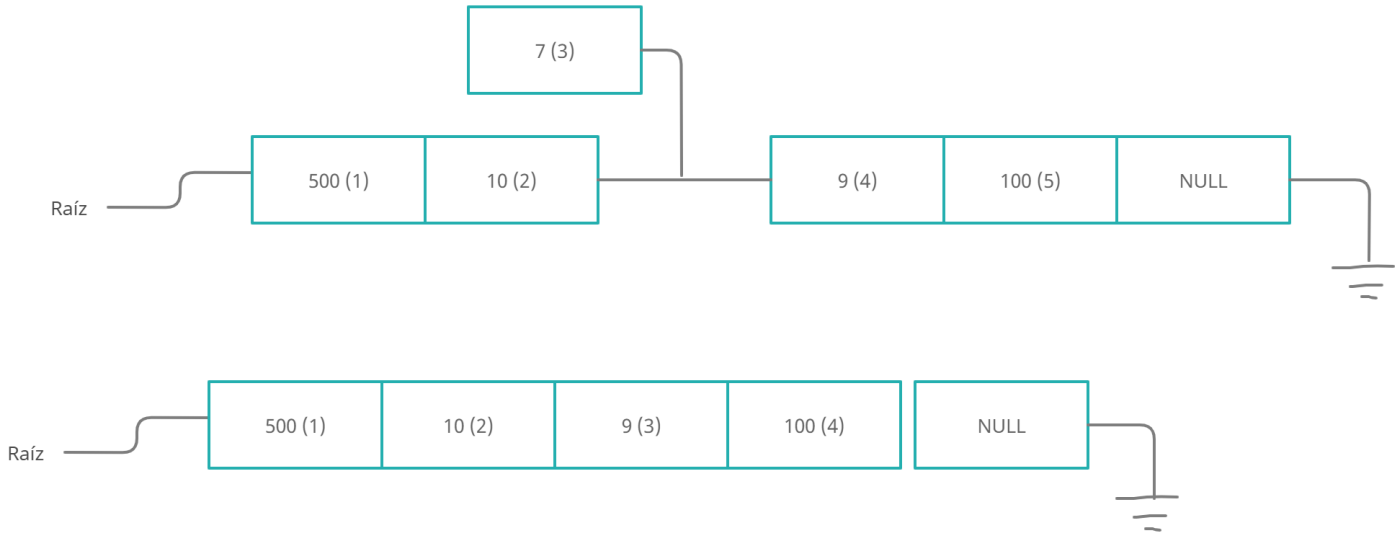

Figura 5: Eliminación del tercer elemento de la lista

Lo siguiente que se hizo fue eliminar el último elemento de la lista mediante la función pop_back, la cual tiene una lógica similar a la eliminación de elementos de la lista, con la diferencia que esta recorre la la lista hasta reconocer mediante NULL, que se ha llegado al último elemento, y si es así, el siguiente elemento del anterior al último nodo, sería NULL y se libera la memoria del que era el último elemento de la lista.

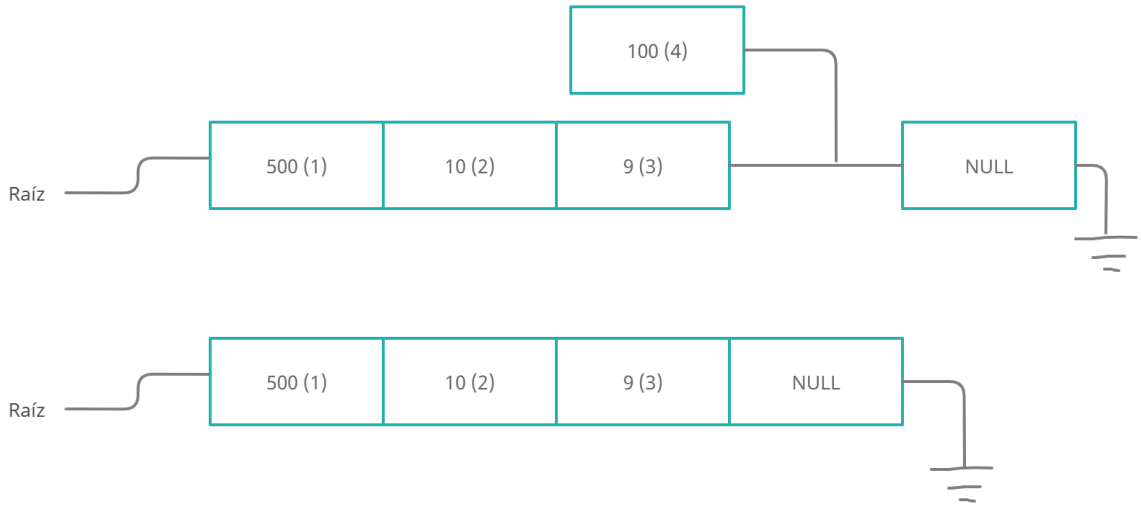

Figura 6: Eliminación del último elemento de la lista

La siguiente acción en el programa principal, es el ordenamiento de elementos mediante el método sort, que recibe la cabeza de la lista y un parámetro de tipo carácter, que si es a, ordena la lista de forma descendente y si es b, de forma ascendente.
Dentro de este método lo que se hizo fue declarar un puntero para reservar memoria dinámica e insertar en él, cada elemento entero de la lista enlazada en este puntero como un arreglo.
Para el ordenamiento, se utilizó el método burbuja para el caso descendente y ascendente, los cuales se implementaron según el carácter ingresado a la función con un condicional. Por último mediante un bucle para la cantidad de elemento en la lista, se imprimió cada elemento de la lista ordenada.

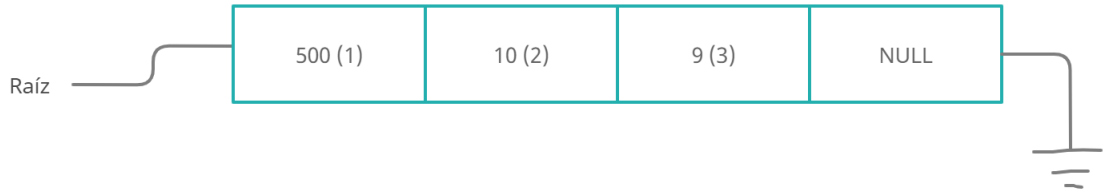

Figura 7: Ordenamiento descendente de la lista

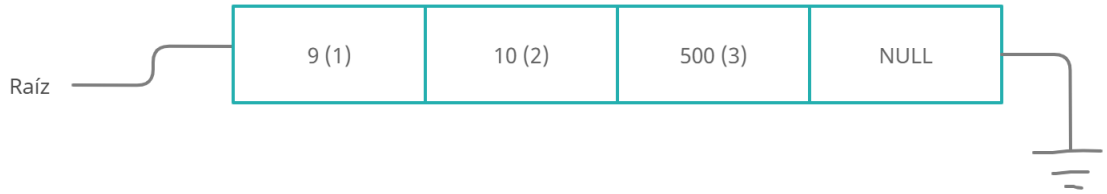

Figura 8: Ordenamiento ascendente de la lista

Para mostrar los resultados en consola de la lista, se utilizó la función printList, que tiene como parámetro la cabeza de la lista y para la escritura y lectura de archivos, se utilizó writeList y readList, donde se indicó en consola explícitamente cuales datos habían sido leídos desde el programa principal y cuales desde el archivo binario con la lista enlazada escrita en él.

0.2.4. Vista de segmentación del programa

Figura 9: Inspección de segmentos del programa

Se observa en la figura 9 que al utilizar el comando size en la terminal de linux para el programa de las lista en lazadas, se despliega la información correspondiente a la segmentación del programa. Primeramente aparece text que contiene las instrucciones del programa, con un tamaño fijo de 767713 bytes.
Seguidamente aparece data, que contiene información de las variables globales inicializadas con un valor de 20772 bytes y bss las no inicializadas con 6848 bytes.

0.2.5. Uso de valgrind

```
juancito@ubuntu:~/Desktop/git/src/src/build$ valgrind --track-origins=yes --dsymutil=yes ./ejecutable
==36238== Memcheck, a memory error detector
==36238== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==36238== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==36238== Command: ./ejecutable
==36238==
==36238== Conditional jump or move depends on uninitialised value(s)
==36238==    at 0x426F00: malloc_hook_ini (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x48313A: _dl_get_origin (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x456314: _dl_non_dynamic_init (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x457F55: __libc_init_first (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x403258: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==  Uninitialised value was created
==36238==    at 0x478C5B: brk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x452AC0: sbrk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4038EC: __libc_setup_tls (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4031DF: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==
==36238== Conditional jump or move depends on uninitialised value(s)
==36238==    at 0x426F91: malloc_hook_ini (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x48313A: _dl_get_origin (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x456314: _dl_non_dynamic_init (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x457F55: __libc_init_first (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x403258: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==  Uninitialised value was created
==36238==    at 0x478C5B: brk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x452AC0: sbrk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4038EC: __libc_setup_tls (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4031DF: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==
==36238== Conditional jump or move depends on uninitialised value(s)
==36238==    at 0x425D8F: tcache_init.part.0 (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x426F9B: malloc_hook_ini (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x48313A: _dl_get_origin (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x456314: _dl_non_dynamic_init (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x457F55: __libc_init_first (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x403258: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==  Uninitialised value was created
==36238==    at 0x478C5B: brk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x452AC0: sbrk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4038EC: __libc_setup_tls (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4031DF: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==
==36238== Conditional jump or move depends on uninitialised value(s)
==36238==    at 0x425339:  int malloc (in /home/juancito/Desktop/git/src/src/build/ejecutable)
```

Figura 10: Uso de valgrind para detección de memory leaks

```
==36238==  Uninitialised value was created by a stack allocation
==36238==    at 0x47C35B: _dl_init_paths (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==
==36238== Conditional jump or move depends on uninitialised value(s)
==36238==    at 0x420456: _IO_default_setbuf (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x41C3EC: _IO_file_setbuf (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x420E65: _IO_cleanup (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x410DF9: __run_exit_handlers (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x410EBF: exit (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4032F6: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==  Uninitialised value was created by a stack allocation
==36238==    at 0x47C35B: _dl_init_paths (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==
==36238== Conditional jump or move depends on uninitialised value(s)
==36238==    at 0x420EB8: _IO_cleanup (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x410DF9: __run_exit_handlers (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x410EBF: exit (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4032F6: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==  Uninitialised value was created
==36238==    at 0x478C5B: brk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x452AC0: sbrk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4038EC: __libc_setup_tls (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4031DF: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==
==36238== Conditional jump or move depends on uninitialised value(s)
==36238==    at 0x420EF8: _IO_cleanup (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x410DF9: __run_exit_handlers (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x410EBF: exit (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4032F6: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==  Uninitialised value was created
==36238==    at 0x478C5B: brk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x452AC0: sbrk (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4038EC: __libc_setup_tls (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==    by 0x4031DF: (below main) (in /home/juancito/Desktop/git/src/src/build/ejecutable)
==36238==
==36238==
==36238== HEAP SUMMARY:
==36238==     in use at exit: 0 bytes in 0 blocks
==36238==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==36238==
==36238== All heap blocks were freed -- no leaks are possible
==36238==
==36238== For lists of detected and suppressed errors, rerun with: -s
==36238== ERROR SUMMARY: 2912 errors from 556 contexts (suppressed: 0 from 0)
juancito@ubuntu:~/Desktop/git/src/src/build$
```

Figura 11: Uso de valgrind para detección de memory leaks

Por medio de la sentencia **valgrind –track-origins=yes –dsymutil=yes ./FILE_NAME** en la terminal de linux, es posible hacer un debug al programa y encontrar errores, además de fugas de memoria que son causados por no liberar memoria reservada, no inicializar punteros o se el valor de retorno de una función de locación de memoria nulo.
En la figura 11, se muestra la sentencia que indica 0 allocs, 0 frees, lo cual confirma que no hay memory leaks por haber reservado memory y no haberla liberado, o bien haber liberado más memoria de la que se reservó.
Todas las funciones fueron llamadas y el programa compila sin errores, salvo que hay advertencia por el uso de otras

variables dentro del programa pero no hace que el programa deje de funcionar.

0.2.6. Árbol de directorios usados en Linux para compilar el programa

Figura 12: Árbol de archivos del programa en Linux

En la figura 12, se muestra el árbol de directorios usados en linux usados para compilar el archivo y obtener el programa principal.
En el directorio build al ejecutar el make, se guarda el archivo ejecutable y el binario con los datos de las lista dentro de él (tipo de dato primitivo). En archivos generados por doxyfile, están los carpetas generadas por el archivo de configuración doxyfile para la documentación del programa en html y latex. En include se encuentra el archivo de encabezado con la estructura para la lista enlazada de la lista y los prototipos de las funciones. En obj, se guardan los archivos tipo objeto al compilar el programa y finalmente en src se encuentran los códigos fuente del programa.

0.3. Conclusiones

1. Para crear la lista dinámica, es necesario, la creación de un primer elemento que sea la cabeza que se defina como un nodo que tiene adentro un elemento entero y como siguiente un NULL para así tener definido un final de la lista.

2. Para la creación de elementos, se reserva memoria dinámica y para la eliminación de elementos en l lista, esta es liberada, por lo tanto es necesario tener en cuanta la liberación de la memoria una vez que no es utilizada para evitar tener errores en memoria y memory leaks.

3. El contenido de los nodos, son datos de tipo (int), por lo que para la escritura de en un archivo binario de los elementos de la lista, es necesario reservar la memoria para la cantidad de enteros dentro de la lista.

4. Las variables inicializadas en e programa, ocupan mayor cantidad de memoria en el programa que las variables no inicializadas.

5. Por medio de valgrind, es posible detectar los memory leaks del programa, sin embargo, este no muestra en que línea está el memory leak y/o error, por ende se utiliza para compilar el flag -l, para mostrar en qué linea está el error.

6. A parte de no liberar la memoria reservada en el programa, otra fuente potencial de errores en el programa, es el declarar punteros y no inicializarlos en algún valor, dado que si no se inicializa en un valor, el puntero se inicializaría en un valor cualquiera y puede ser que al ejecutar el programa, esa dirección de memoria ya fue ocupada y podría ser un error potencial, por ende, una solución, es inicializarlos en NULL.

7. Al trabajar con punteros y estructuras, es importante considerar el hecho que el operador de asignación "..^{en} la estructura, tiene mayor precedencia que el operador "*.^{en} contexto de ejecución, por lo que para el correcto uso, es necesario el uso de paréntesis o bien el operador "->". En el caso de puntero dobles usar (*variable)->valor.

Bibliografía

[1] UCM3. (s.f). *La función printf*.Recuperado el 01 de febrero de 2021. Desde, http://www.it.uc3m.es/pbasanta/asng/course_notes/input_output_printf_es.html

[2] Arquitectura de Sistemas UC3M. (s.f) *Listas enlazadas en C*. Recuperado el 28 de febrero de 2021. Desde, http://www.it.uc3m.es/pbasanta/asng/course_notes/ch16.html

[3] Universidad de Concepción. (202) *Manejo de archivos en C++ 1*. Recuperado el 28 de febrero de 2021. Desde, https://w3.ual.es/~abecerra/ID/archivos.pdf

[4] Bonvallet, Roberto. (202) *El lenguaje C* . Recuperado el 28 de febrero de 2021. Desde, https://www.fing.edu.uy/tecnoinf/mvd/cursos/prinprog/material/teo/prinprog-teorico08.pdf