EIE

Escuela de
Ingeniería Eléctrica

UNIVERSIDAD DE COSTA RICA

UNIVERSIDAD DE COSTA RICA

FACULTAD DE INGENIERIA

ESCUELA DE INGENIERIA ELÉCTRICA

PROGRAMACIÓN BAJO PLATAFORMAS ABIERTAS

LABORATORIO 6:
LISTAS ENLAZADAS

ESTUDIANTE:
JESÚS ZÚÑIGA MÉNDEZ

PROFESOR:
RICARDO ROMÁN BRENES; M. SC.

I CICLO 2019

Índice

1. **Introducción** 2
2. **includes.h** 3
3. **posicion.h** 4
4. **lista.h** 5
5. **posicion.c** 6
6. **lista.c** 7
7. **main.c** 10
8. **Conclusión** 14

Índice de figuras

1. Introducción

Para este laboratorio el objetivo fue crear un programa que pudiera manipular listas enlazadas, a continuación se adjunta el código escrito en C con el que se soluciono el problema, en el código se encuentra la documentación necesaria para comprender que lógica que siguen las funciones presentes en el programa.

2. includes.h

```
#ifndef INCLUDES_H
#define INCLUDES_H
#include <stdio.h>
#include <stdlib.h>

typedef struct _posicion posicion;
typedef struct _lista lista;

#include "./posicion.h"
#include "./lista.h"


#endif
```

3. posicion.h

```
#ifndef POSICION1_H
#define POSICION1_H

#include "includes.h"
    typedef
        struct _posicion
        {
            int d;
            struct _posicion* siguiente;
        }
    posicion;
    posicion* Siguiente(posicion*);
    posicion* Anterior(posicion*, lista*);
    posicion* CrearPosision(int);
    void EliminarPosicion(posicion*);

#endif
```

4. lista.h

```
#ifndef LISTA_H
#define LISTA_H
#include "includes.h"

    typedef
        struct _lista
        {
            unsigned int items;
            posicion* primero;
        }
    lista;

    lista** Estirar(lista**, int, int);
    lista* CrearLista();
    void ImprimirLista(lista*);
    void EliminarLista(lista*);
    void Vaciar(lista*);
    void AgregarElemento(lista*, int);
    void EliminarElemento(lista*, int);
    posicion* BuscarDato(lista*, int);
    int BuscarK(lista*, posicion*);
#endif
```

5. posicion.c

```
//**
 * @file posicion.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo que contiene las funciones para manipular las posiciones de una ñista
 */
#include "../include/includes.h"
/**
 * @brief Funcion que devuelve la posicion siguiente de una lista
 * @param p posicion inicial
 */
posicion* Siguiente(posicion* p){
    posicion* Np = p->siguiente;
    return Np;
}
/**
 * @brief Funcion que devuelve la posicion anterior de una lista
 * @param p posicion inicial
 * @param ls lista en la que se va a buscar
 */
posicion* Anterior(posicion* p, lista* ls){
    posicion* anterior=ls->primero;
    posicion* actual = ls-> primero;
    if (actual != 0x0){
        for (int i=1; i<= ls->items; i++){
            if (actual == p){
                i=ls->items;
            }else
            {
                anterior = actual;
                actual = actual->siguiente;
            }

        }
    }
    return anterior;
}
/**
 * @brief Funcion que crea una posicion vacia
 * @param dato Es el dato que se ve a agregar
 */
posicion* CrearPosision(int dato){
    posicion* p = (posicion*) malloc(sizeof(posicion));
    p->d= dato;
    return p;
}
/**
 * @brief Funcion que elimina una poicion
 * @param p es el posicion que se ve a eliminar
 */
void EliminarPosicion(posicion* p){
    free(p);
}
```

6. lista.c

```
/**
 * @file lista.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo que contiene las funciones para manipular listas
 */
#include "../include/includes.h"

/**
 * @brief Funcion que agranda la lista que contienen las demas listas
 * @return listaViejo es la lista que se acaba de crear con un espacio mas
 * @param listaAnterior es la lista original
 * @param tamanioViejo es la cantidad de listas que existian antes de estirar
 * @param TamanioNuevo es el nuevo numero de listas
 */

lista** Estirar(lista** listaAnterior, int tamanioViejo, int tamanioNuevo){
    lista** nuevaLista = (lista**) malloc(tamanioNuevo * sizeof(lista));
    for(int i = 0; i < tamanioViejo; i++)
    {
        nuevaLista[i] = listaAnterior[i];
    }
    free(listaAnterior);
    listaAnterior = nuevaLista;
    return listaAnterior;
}


/**
 * @brief Funcion que crea una lista vacia
 * @return ls es la lista que se acaba de crear;
 */

lista* CrearLista(){
    lista* ls = (lista*) malloc (sizeof(lista));
    ls->items = 0;
    ls->primero = 0x0;
    return ls;
}


/**
 * @brief Funcion que elimina una lista
 * @param ls lista con la que se va a trabajar
 */

void EliminarLista(lista* ls){
    posicion* q;
    posicion* p = ls->primero;
    for (int i = 0; i < ls->items; i++){
```

```
        q = Siguiente(p);
        EliminarPosicion(p);
        p=q;
    }
    free(ls);
}
/**
 * @brief Funcion que llena una lista de ceros
 * @param ls lista con la que se va a trabajar
 */
void Vaciar(lista* ls){
    posicion* p = ls->primero;
    for (int i = 0; i< ls->items; i++){
        p->d = 0;
        p = Siguiente(p);
    }

}
/**
 * @brief Funcion que agrega un elemento a la lista
 * @param ls lista con la que se va a trabajar
 * @param elemento elemento que se va a agregar
 */
void AgregarElemento(lista* ls, int elemnto){
    posicion* nuevaP= CrearPosision(elemnto);
    posicion* p = ls -> primero;
    if (p == 0x0){
        ls -> primero = nuevaP;
        nuevaP ->siguiente = 0x0;
        ls ->items++;
    }else{
        for (int i = 1; i< ls->items; i++){
            p = Siguiente(p);
        }
        p->siguiente= nuevaP;
        nuevaP ->siguiente = 0x0;
        ls->items++;
    }


}
/**
 * @brief Funcion que elimina un elemnto de la lista
 * @param ls lista con la que se va a trabajar
 * @param elemento elemento que se va a quitar
 */
void EliminarElemento(lista* ls, int elemento){
    posicion* p = ls -> primero;
    posicion* anterior = Anterior(p,ls);
    int contador = 1;
    if (p == 0x0){
        printf("La lista esta vacia, no hay elemntos que eliminar\n");
    }else{
        while (contador <= ls->items){
```

```
            printf("contador %d\tvalor %d\tls->items %d\n",contador,p->d,ls->items);
            if (p->d == elemento){
                if (contador==1){
                    printf("borre al inicio\n");
                    ls->primero = Siguiente(p);
                    EliminarPosicion(p);
                    p=ls->primero;
                    anterior = Anterior(p,ls);
                    ls->items--;
                }else{
                    printf("borre al medio\n");
                    anterior->siguiente = Siguiente(p);
                    EliminarPosicion(p);
                    p=anterior->siguiente;
                    ls->items--;
                    if (ls->items < 1){
                        ls->primero = 0x0;
                    }
                }
            }else{
                contador++;
                p = Siguiente(p);
                anterior=Anterior(p,ls);
            }
        }
    }
}
/**
 * @brief Funcion que imprime una lista
 * @param ls lista que se va a imprimir
 */
void ImprimirLista(lista* ls){
    posicion* p = ls->primero;
    for (int i = 0; i < ls->items; i++){
        printf("%d  ", p->d);
        p = Siguiente(p);
   }
   printf("\n");
}
/**
 * @brief Funcion que busca un dato en la lista
 * @param ls lista donde se va a buscar
 * @param dato es el dato que se va a buscar
 */
posicion* BuscarDato(lista* ls, int dato){
    int contadorDatos=0;
    posicion* p = ls->primero;
    posicion* laPosicion = 0x0;
    for (int i = 1; i <= ls->items; i++){
        if (p->d == dato){
            contadorDatos++;
            laPosicion = p;
        }
        p = Siguiente(p);
```

```
    }
    p = ls->primero;
    if (contadorDatos>1){
        printf("Se encontraron %d coincidencias\n",contadorDatos);
         for (int i = 1; i <= ls->items; i++){
             if (p->d == dato){
                 printf("Se encontraron coincidencias en la direccion %p\n",p);
             }
             p = Siguiente(p);
         }
         laPosicion = 0x0;
    }
    return laPosicion;
}
/**
 * @brief Funcion que busca una posicion dentro de la lista
 * @param ls lista donde se va a buscar
 * @param laPosicion es la posicion que se va a buscar
 */
int BuscarK(lista* ls,posicion* laPosicion){
    int dato=0;
    posicion* p = ls->primero;
    for (int i = 1; i <= ls->items; i++){
        if (p==laPosicion){
            dato=p->d;
        }
        p = Siguiente(p);
    }
    return dato;
}
```

7. main.c

```
/**
 * @file main.c
 * @author Jesus Zuñiga Mendez
 * @brief Archivo pricipal, laboratorio que trata sobre listas enlazadas
 * @version 1.0
 * @date 20 de junio de 2019
 * @copyright Copyleft (l) 2019
 */
#include "./include/includes.h"

/**
 * @brief Función que imprime una bienvenida para el usuario
 */
void Bienvenida(){
    printf("  ____   _                                _      _          _  _  _ \n |  _ \\
    printf("\n");
    printf("\n");
}
```

```
/**
 * @brief Funcion que imprime el menu
 * @return seleccion Es la opcion escogida por el usuario
 */
int Menu(){
    printf("Digite el numero de la opcion que quiere realizar.\n\n\n");
    printf("1. Crear una lista.\t\n");
    printf("2. Eliminar lista.\t\n");
    printf("3. Vaciar lista.\t\n");
    printf("4. Agregar elemento.\t\n");
    printf("5. Eliminar elemento.\t\n");
    printf("6. Imprimir lista.\t\n");
    printf("7. Buscar dato\t\n");
    printf("8. Buscar posicion.\t\n");
    printf("0. SALIR.\t\n\n\n");
    int seleccion = 0;
    int s = scanf("%d",&seleccion);
    return seleccion;
}

/**
 * @brief Funcion que libera la memoria utilizada por la lista que contiene las demas lista
 * @param listaListas es la lista que contiene las demas listas
 * @param cantidad es la cantidad de listas que creo el usuario
 */
void LiberarListaListas(lista** listaListas, int cantidad){
    for (int i = 0; i < cantidad; i++){
        free(listaListas[i]);
    }
    free(listaListas);
}


/**
 * @brief Funcion Principal, contiene el codigo que ejecuta las demas acciones
 */
int main(int argc, char** argv){
    Bienvenida();
    int cantidadListas = 1;
    int seleccion = 0;
    int s = 0;
    int dato =0;
    posicion* laPosicion;
    //lista que contendra las demas listas
    lista** listaListas = (lista**) malloc (1*sizeof(lista));
    int opcion = 0;
    lista* ls = NULL;
    do{
        opcion = Menu();
        switch (opcion)
        {
        case 1:
            listaListas = Estirar(listaListas, cantidadListas, (cantidadListas + 1));
            listaListas[cantidadListas-1] = CrearLista ();
```

```
    printf("Lista %d creada,  Consultela con este identificador:\n", cantidadListas
    cantidadListas++;
    break;
case 2:
    printf("¿Cual lista desea eliminar?:\n");
    s = scanf("%d",&seleccion);
    if (listaListas[seleccion-1]==NULL){
        printf("La lista no existe\n");
    }else{
        EliminarLista(listaListas[seleccion-1]);
        listaListas[seleccion-1] = NULL;
        printf("Lista Eliminada satisfactoriamente.\n");
    }
    break;
case 3:
    printf("¿Cual lista desea vaciar?:\n");
    s = scanf("%d",&seleccion);
    if (listaListas[seleccion-1]==NULL){
        printf("La lista no existe\n");
    }else{
        Vaciar(listaListas[seleccion-1]);
        printf("Lista Vaciada satisfactoriamente.\n");
    }
    break;
case 4:
    printf("¿A cual lista le desea agregar elementos?:\n");
    s = scanf("%d",&seleccion);
    if (listaListas[seleccion-1]==NULL){
        printf("La lista no existe\n");
    }else{
        printf("Digite el numero que desea agregar\n");
        s = scanf("%d", &dato);
        AgregarElemento(listaListas[seleccion-1],dato);
        printf("Lista actualizada correctamente.\n");
    }
    break;
case 5:
    printf("¿A cual lista le desea eliminar elementos?:\n");
    s = scanf("%d",&seleccion);
    if (listaListas[seleccion-1]==NULL){
        printf("La lista no existe\n");
    }else{
        printf("Digite el elemnto que desea eliminar\n");
        s = scanf("%d", &dato);
        EliminarElemento(listaListas[seleccion-1],dato);
        printf("Lista actualizada correctamente.\n");
    }
    break;
case 6:
    printf("¿Cual lista desea Imprimir?:\n");
    s = scanf("%d",&seleccion);
    if (listaListas[seleccion-1]==NULL){
        printf("La lista no existe\n");
    }else{
```

```
                ImprimirLista(listaListas[seleccion-1]);
            }
            break;
        case 7:
            printf("¿En cual lista desea buscar elementos?:\n");
            s = scanf("%d",&seleccion);
            if (listaListas[seleccion-1]==NULL){
                printf("La lista no existe\n");
            }else{
                printf("Digite el elemnto que desea buscar\n");
                s = scanf("%d", &dato);
                laPosicion = BuscarDato(listaListas[seleccion-1],dato);
                if (laPosicion != 0x0){
                    printf("Lista revisada correctamente el dato se encuentra en la posicio
                }else{
                    printf("Lista revisada correctamente No se encontraron coincidencias\n"
                }

            }
            break;
        case 8:
            printf("¿En cual lista desea buscar la posicion?:\n");
            s = scanf("%d",&seleccion);
            if (listaListas[seleccion-1]==NULL){
                printf("La lista no existe\n");
            }else{
                printf("Digite la posicion que desea buscar\n");
                s = scanf("%p", &laPosicion);
                printf("Lei esto %p\n",laPosicion);
                dato = 0;
                dato = BuscarK(listaListas[seleccion-1],laPosicion);
                if (dato != 0){
                    printf("Lista revisada correctamente en la posicion se encuentra el ele
                }else{
                    printf("Lista revisada correctamente No se encontraron coincidencias\n"
                }

            }
            break;
        case 0:
            printf("cero\n");
            break;
        default:
            printf("seleccione un numero correcto\n");
            break;
        }
    }while (opcion !=0);
    LiberarListaListas(listaListas, (cantidadListas-1));
    return 0;
}
```

8. Conclusión

Se concluye que las listas enlazadas permiten manipular datos de una forma sencilla, ya que con implementar ciertas funciones como las que están en el archivo posicion.c se puede recorrer las listas sin problemas, ademas debido a que se debió utilizar varios archivos de cabecera se aprendió el uso de guards para los headers ademas de realizar forward declaration para evitar que las estructuras no fueran reconocidas.