Trabajo practico N°4
"LinkedList"

Programación I – Laboratorio I.
Tecnicatura Superior en Programación.
UTN-FRA

Autores: *Mg. Mauricio Dávila*

Revisores: *Esp. Ernesto Gigliotti*

Versión : 1

Índice de contenido

1 Objetivo

El objetivo del siguiente trabajo es que el alumno sea capaz de demostrar que puede integrar lo aprendido durante la cursada en un caso real.

Los conocimientos necesarios para la realización del TP son los siguientes:

- Manejo de punteros.
- Manejo de arrays.
- Manejo de estructuras.
- Manejo de memoria dinámica.

1.1 Etapas del trabajo

Etapa 1: Se deberá desarrollar una biblioteca LinkedList.c y LinkedList.h la cual contendrá el tipo de dato LinkedList, tal que cumpla con la especificación del documento, con las funciones mínimas requeridas.

Etapa 2: Realizar una aplicación que dé uso del **LinkedList** (usando todas las funciones) y que permita interactuar con estructuras de datos almacenadas en archivos.

1.2 Condiciones de Entrega

El trabajo práctico es de carácter individual y cada una de sus entregas (sin excepción) deben ser enviadas en la fecha establecida por los docentes. Para reducir el uso de papel, la modalidad de las entregas será en forma totalmente digital utilizando como medio un repositorio de github.com, el cual será informado al docente a través de un mensaje en el campus. Las devolución de los docentes será realizado por el mismo medio. En la misma, se indicarán las observaciones pertinentes, pudiendo solicitarse la realización de algunos cambios si fuera necesario, además de informar si la entrega está aprobada o no.

1.3 Condiciones de Aprobación

Se deberá entregar un proyecto de código ANSI C el cual estará compuesto de un programa que utilice la biblioteca **LinkedList** en su totalidad, el mismo deberá contar como mínimo con las funciones obligatorias, todas ellas con su respectiva documentación, y un programa que utilice de manera integral la biblioteca. En la fecha del segundo parcial se realizará una defensa oral del trabajo por parte del alumno a efectos de determinar si se encuentra o no en condiciones de rendir examen final.

2 Biblioteca LinkedList

El LinkedList es una estructura que permite almacenar datos en memoria de forma similar a los Arrays, con la ventaja de que el número de elementos que almacena es dinámico, es decir, que no es necesario declarar su tamaño como pasa con los Arrays. Los LinkedList nos permiten añadir, eliminar y modificar elementos de forma transparente para el programador.

```
#ifndef __LINKEDLIST
#define __LINKEDLIST
struct Node
{
	void* pElement;
	struct Node* pNextNode;
}typedef Node;

struct LinkedList
{
	Node* pFirstNode;
	int size;
}typedef LinkedList;
#endif
```

Cada función de la biblioteca cuenta con un Test unitario asociado mediante el cual se podrá verificar el correcto funcionamiento de la misma.

```
startTesting(1); // ll_newLinkedList
startTesting(2); // ll_len
startTesting(3); // getNode - test_getNode
startTesting(4); // addNode - test_addNode
startTesting(5); // ll_add
startTesting(6); // ll_get
startTesting(7); // ll_set
startTesting(8); // ll_remove
startTesting(9); // ll_clear
startTesting(10); // ll_deleteLinkedList
startTesting(11); // ll_indexOf
startTesting(12); // ll_isEmpty
startTesting(13); // ll_push
startTesting(14); // ll_pop
startTesting(15); // ll_contains
startTesting(16); // ll_containsAll
startTesting(17); // ll_subList
startTesting(18); // ll_clone
startTesting(19); // ll_sort
```

2.1 Función ll_newLinkedList

Crea y retorna un nuevo LinkedList. Es el constructor, ya que en él daremos valores iniciales a las variables y asignaremos las funciones a sus punteros.

```
LinkedList* ll_newLinkedList(void)
{
    //.......
}
```

Ejemplo uso:

```
LinkedList* lista;
lista = ll_newLinkedList();
```

Ejemplo de uso del test:

```
startTesting(1);
```

2.2 Función ll_len

Retorna el tamaño del LinkedList. Verificando que el puntero this sea distinto de NULL. Si la verificación falla la función retorna (-1) y si tiene éxito retorna la longitud del array.

```
int ll_len(LinkedList* this)
{
    //.......
}
```

Ejemplo uso:

```
longitud = ll_len(lista);
```

Ejemplo de uso del test:

```
startTesting(2);
```

2.3 Función getNode

Retorna un puntero al nodo que se encuentra en el índice especificado. Verificando que el puntero this sea distinto de NULL y que index sea positivo e inferior al tamaño del array. Si la verificación falla la función retorna (NULL) y si tiene éxito retorna el puntero al nodo.

```
static Node* getNode(LinkedList* this , int nodeIndex)
{
    //.......
}
```

Ejemplo uso:

```
Node* nodo;
nodo = ll_getNode(lista,5);
```

Ejemplo de uso del test:

```
startTesting(3);
```

2.4 Función ll_push

Agrega un nodo en la posición indexNode. Verificando que el puntero this sea distinto de NULL y que index sea positivo e inferior al tamaño del array. Si la verificación falla la función retorna (-1) y si tiene éxito (0).

```
static int addNode(LinkedList* this, int nodeIndex, void* pElement)
{
    //.......
}
```

Ejemplo uso:

```
Nodo Nodo;
r = addNode(lista,6,&Nodo);
```

Ejemplo de uso del test:

```
startTesting(4);
```

2.5 Función ll_add

Agrega un elemento al final de LinkedList. Verificando que el puntero this sea distinto de NULL. Si la verificación falla la función retorna (-1) y si tiene éxito (0).

```
int ll_add(LinkedList* this,void* pElement)
{
    //.......
}
```

Ejemplo uso:

```
Persona auxPersona;
r = ll_add(lista,&auxPersona);
```

Ejemplo de uso del test:

```
startTesting(5);
```

2.6 Función ll_get

Retorna un puntero al elemento que se encuentra en el índice especificado. Verificando que el puntero this sea distinto de NULL y que index sea positivo e inferior al tamaño del array. Si la verificación falla la función retorna (NULL) y si tiene éxito retorna el elemento.

```
void* ll_get(LinkedList* this , int index);
{
    //.......
}
```

Ejemplo uso:

```
Persona* elemento;
elemento = (Persona*)ll_get(lista,5);
```

Ejemplo de uso del test:

```
startTesting(6);
```

2.7 Función ll_set

Inserta un elemento en el LinkedList, en el índice especificado. Verificando que el puntero this sea distinto de NULL y que index sea positivo e inferior al tamaño del array. Si la verificación falla la función retorna (-1) y si tiene éxito (0).

```
int ll_set(LinkedList* this, int index,void* pElement)
{
    //.......
}
```

Ejemplo uso:

```
Persona auxPersona;
r = ll_set(lista,4,&auxPersona);
```

Ejemplo de uso del test:

```
startTesting(7);
```

2.8 Función ll_remove

Elimina un elemento del LinkedList, en el índice especificado. Verificando que el puntero this sea distinto de NULL y que index sea positivo e inferior al tamaño del array. Si la verificación falla la función retorna (-1) y si tiene éxito (0).

```
int ll_remove(LinkedList* this,int index);
{
    //.......
}
```

Ejemplo uso:

```
r = ll_remove(lista,5);
```

Ejemplo de uso del test:

```
startTesting(8);
```

2.9 Función ll_clear

Borra todos los elementos de LinkedList. Verificando que el puntero this sea distinto de NULL. Si la verificación falla la función retorna (-1) y si tiene éxito (0).

```
int ll_clear(LinkedList* this)
{
    //.......
}
```

Ejemplo uso:

```
r = ll_clear(lista);
```

Ejemplo de uso del test:

```
startTesting(9);
```

2.10 Función ll_deleteLinkedList

Elimina el LinkedList . Verificando que el puntero this sea distinto de NULL. Si la verificación falla la función retorna (-1), si esta vacío (1) y si contiene elementos (0).

```
int ll_deleteLinkedList(LinkedList* this)
{
    //.......
}
```

Ejemplo uso:

```
r = ll_deleteLinkedList(lista);
```

Ejemplo de uso del test:

```
startTesting(10);
```

2.11 Función ll_indexOf

Retorna el índice de la primera aparición de un elemento (element) en el LinkedList. Verificando que el puntero this sea distinto de NULL. Si la verificación falla o no encuentra el elemento la función retorna (-1) y si encuentra el elemento retorna su índice.

```
int ll_indexOf(LinkedList* this, void* element)
{
    //.......
}
```

Ejemplo uso:

```
r = ll_indexOf(lista,&auxPersona))
```

Ejemplo de uso del test:

```
startTesting(11);
```

2.12 Función ll_isEmpty

Retorna cero si contiene elementos y uno si no los tiene. Verificando que el puntero this sea distinto de NULL. Si la verificación falla la función retorna (-1), si esta vacío (1) y si contiene elementos (0).

```
int ll_isEmpty(LinkedList* this)
{
    //.......
}
```

Ejemplo uso:

```
if(ll_isEmpty(lista))
      printf ("Esta vacío");
```

Ejemplo de uso del test:

```
startTesting(12);
```

2.13 Función ll_push

Desplaza los elementos e inserta en la posición index. Verificando que el puntero this sea distinto de NULL y que index sea positivo e inferior al tamaño del array. Si la verificación falla la función retorna (-1) y si tiene éxito (0).

```
int ll_push(LinkedList* this, int index, void* pElement)
{
    //.......
}
```

Ejemplo uso:

```
Persona auxPersona;
r = ll_set(lista,6,&auxPersona);
```

Ejemplo de uso del test:

```
startTesting(13);
```

2.14 Función ll_pop

Retorna un puntero al elemento que se encuentra en el índice especificado y luego lo elimina de la lista. Verificando que el puntero this sea distinto de NULL y que index sea positivo e inferior al tamaño del array. Si la verificación falla la función retorna (NULL) y si tiene éxito retorna el elemento.

```
void* ll_pop(LinkedList* this , int index);
{
    //.......
}
```

Ejemplo uso:

```
Persona* elemento;
elemento = (Persona*)ll_pop(lista,5);
```

Ejemplo de uso del test:

```
startTesting(14);
```

2.15 Función ll_contains

Comprueba si existe el elemento que se le pasa como parámetro. Verificando que tanto el puntero this sea distintos de NULL. Si la verificación falla la función retorna (-1) , si encuentra el elemento (1) y si no lo encuentra (0).

```
int ll_contains(LinkedList* this, void* pElement)
{
    //.......
}
```

Ejemplo uso:

```
if(ll_contains(lista,&auxPersona))
      printf ("SI");
```

Ejemplo de uso del test:

```
startTesting(15);
```

2.16 Función ll_containsAll

Comprueba si los elementos pasados son contenidos por el LinkedList. Verificando que tanto el puntero this como pList2 sean distintos de NULL. Si la verificación falla o no encuentra el elemento la función retorna (-1), si las listas difieren (0) y si ambas listas son iguales retorna (1).

```
int ll_containsAll(LinkedList* this,LinkedList* this2)
{
    //.......
}
```

Ejemplo uso:

```
      if(ll_containsAll(lista_A,lista_B)))
            printf ("Contienen los mismos elementos");
```

Ejemplo de uso del test:

```
      startTesting(16);
```

2.17 Función ll_sublist

Retorna un nuevo LinkedList con el subconjunto de elementos. Verificando que el puntero this sea distinto de NULL y que tanto el indice 'from' como 'to' sean positivos e inferiores al tamaño del array. Si la verificación falla la función retorna (NULL) y si tiene éxito retorna el nuevo array.

```
LinkedList* ll_subList(LinkedList* this,int from,int to)
{
    //.......
}
```

Ejemplo uso:

```
      Persona* elemento;
      elemento = (Persona*)ll_pop(lista,5);
```

Ejemplo de uso del test:

```
      startTesting(17);
```

2.18 Función ll_clone

Retorna un nuevo LinkedList copia del LinkedList original. Verificando que el puntero this sea distinto de NULL. Si la verificación falla la función retorna (NULL) y si tiene éxito retorna el nuevo array.

```
LinkedList* ll_clone(LinkedList* this)
{
    //.......
}
```

Ejemplo uso:

```
      LinkedList* arrayClon;
      arrayClon = ll_clone(lista);
```

Ejemplo de uso del test:

```
      startTesting(18);
```

2.19 Función ll_sort

Ordena los elementos del array recibiendo como parámetro la función que sera la encargada de determinar que elemento es mas grande que otro y si se debe ordenar de manera ascendente o descendente. Verificando que tanto el puntero this como el puntero a la funcion pFunc sean distintos de NULL. Si la verificación falla (-1) caso contrario retorna (1).

```
int ll_sort(LinkedList* this, int (*pFunc)(void* ,void*), int order)
{
    //.......
}
```

Ejemplo de la función de comparación:

```
int comparaPersonas(void* pPersonA,void* pPersonB)
{
      if(((Persona*)pPersonA)->edad > ((Persona*)pPersonB)->edad)
      {
      return 1;
      }
      if(((Persona*)pPersonA)->edad < ((Persona*)pPersonB)->edad)
      {
            return -1;
      }
      return 0;
}
```

Ejemplo de uso

```
r = ll_sort(lista, comparaPersonas,1);
```

Ejemplo de uso del test:

```
startTesting(19);
```