# Práctica 8: Listas doblemente ligadas

Estructuras de Datos y Algoritmos I

Autor: José Mauricio Matamoros de Maria y Campos

## 1. Objetivo

El alumno revisará las definiciones, características, procedimientos y ejemplos de la estructura lineal *lista doblemente ligada* tanto en sus versiones de extremos abiertos como circulares para poder comprenderlas a fin de ser capaz de implementarlas.

## 2. Material

Se asume que el alumno cuenta con un una computadora con arquitectura Intel x86 o compatible, sistema operativo Linux/Unix, y compilador `gcc` instalado.

## 3. Instrucciones

1. Lea detenidamente los antecedentes teóricos sobre la construcción de listas doblemente ligadas y listas doblemente ligadas circulares Sección 4.
2. Realice cada una de las actividades detalladas en la Sección 5 apoyándose en el código de ejemplo del Apéndice A y responda las preguntas de las mismas.
3. Finalmente, responda el cuestionario de la Sección 6.

## 4. Antecedentes

Una lista enlazada o ligada es una estructura de datos en la que los objetos se organizan en orden lineal. Sin embargo, a diferencia de un arrglo en el que el orden está determinado por los índices, el orden en una lista ligada lo determina un apuntador en cada eslabón u objeto. Las listas ligadas ofrecen una representación simple y flexible para conjuntos dinámicos que admiten (aunque no necesariamente de manera eficiente) todas las operaciones simples (búsqueda, inserción, eliminación, mínimo, máximo, sucesor y predcesor).

Como se muestra en la Figura 1, cada elemento de una lista L doblemente ligada es un objeto con una llave-valor y dos apuntadores: $x.next$ o elemento siguiente y $x.prev$ o elemento anterior. El objeto también puede contener adicionales o bien tener como llave-valor un apuntador a una estructura más compleja u objeto. Dado un elemento x en la lista, $x.next$ apunta a su sucesor en la lista vinculada, y $x.prev$ apunta a su predecesor. Si $x.prev = NIL$, el elemento $x$ no tiene predecesor y, por tanto, es el primer elemento (o cabeza) de la lista. Si $x.next = NIL$, el elemento $x$ no tiene sucesor y, por lo tanto, es el último elemento (o cola) de la lista. Un atributo $L.head$ apunta al primer elemento de la lista. Si $L : head = NIL$, la lista está vacía.

Una lista puede tener una de varias formas. Puede ser ligada sencilla o doblemente ligada, puede ser ordenada o no, y puede ser circular o no. Si una lista es sencillamente ligada se omite el apuntador prev en cada elemento. Si la lista es ordenada, el orden lineal de la lista corresponde al orden lineal de las llaves-valor almacenadas en los elementos de la lista. En este caso, el elemento mínimo es entonces el encabezado de la lista y el elemento máximo es la cola. Si la lista no es ordenada, los elementos pueden aparecer en cualquier orden. En una lista circular, el apuntador *prev* de la cabeza de la lista apunta a la cola y el apuntador *next* de la cola de la lista apunta a la cabeza. Se puede pensar en una lista circular como un anillo de elementos.

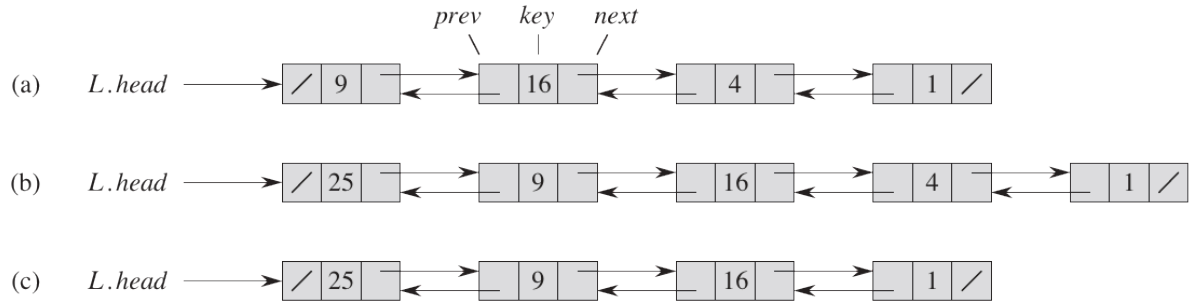


Figura 1: **(a)** Una lista $L$ doblemente ligada que representa el conjunto dinámico $L = \{1, 4, 9, 16\}$.Cada elemento de la lista es un objeto con atributos para la clave y punteros (mostrados por flechas) al objeto anterior y siguiente. El siguiente atributo de la cola y el atributo *prev* de la cabeza son ambos NIL, indicado por una barra diagonal. El atributo $L.head$ apunta a la cabeza. **(b)** Tras la ejecución de List-Insert$(L, x)$, donde $x.key = 25$, la lista ligada tiene un nuevo objeto con llave-valor 25 como nueva cabeza. Este nuevo objeto apunta a la cabeza anterior con llave-valor 9. **(c)** El resultado de la siguiente llamada List-Delete$(L, x)$, donde x apunta al objeto con la llave-valor 4.

# 5. Desarrollo de la práctica

Modifique el código del Apéndice A.2 para generar un programa que permita recorrer la lista generada en un bucle infinito en ambos sentidos.
¿Qué salida produce el programa? [1 punto]: _____

¿Cómo se implementó la función `insert`? Anote y explique su código a continuación [2 puntos]:

¿Cómo se implementó la función `append`? Anote y explique su código a continuación [2 puntos]:

¿Cómo se implementó la función `delete`? Anote y explique su código a continuación [2 puntos]:

## 6. Cuestionario

1. ¿Qué estructura de datos requiere el programa? Explique [1 punto]: _____

2. ¿Cuál es la diferencia entre una lista ligada y una lista doblemente ligada? Explique [1 punto]: _____

3. ¿Cuál es la diferencia entre una lista doblemente ligada y una lista doblemente ligada circular? Explique [1 punto]: _____

# A. Código de ejemplo

## A.1. Archivo **src/Makefile**

src/Makefile

```
CC       = gcc
CFLAGS   = -Wall -O0
SILENT   = @
PROGRAMS = prog1
.PHONY: all clean $(PROGRAMS)

all: $(PROGRAMS)

$(PROGRAMS): %: %.c
  $(SILENT) $(CC) $(CFLAGS) -o $@ $<

clean:
  rm -f *.o
```

## A.2. Archivo `src/prog1.c`

src/prog1.c

```c
#include <stdio.h>
#include <stdlib.h>
struct tnode{
  int value;            //key
  struct tnode* next;
  struct tnode* prev;
} typedef node;

struct{
  node* first;
  int count;
} typedef list;
void append(list* l, int x);
void insert(list* l, int ix, int x);
void delete(list* l, int ix);
int list_size(list* l);
void list_print_segment(list* l, int offset, int count);
list* list_init();
node* node_init(int value);
node* get_node(list* l, int ix);
int main(void){
  int x;
  // 1. Inicializar estructura de datos
  list *l = list_init();
  // 2. Llenar la lista
  int n = 1 + rand() % 100;
  for(size_t i = 0; i < n; ++i){
    append(l, 1 + rand() % 1000);
  }

  printf("List size: %d\n", l->count);
  delete(l, rand() % list_size(l));
  delete(l, rand() % list_size(l));
  delete(l, rand() % list_size(l));
  printf("List size: %d\n", l->count);
  x = 1 + rand() % 100;
  insert(l, rand() % list_size(l), x);
  x = 1 + rand() % 100;
  insert(l, rand() % list_size(l), x);
  printf("List size: %d\n", l->count);

  // 3. Imprimir la lista
  list_print_segment(l, 0, 5);
  list_print_segment(l, 10, 20);
  list_print_segment(l, -5, 0);
  list_print_segment(l, -5, -5);
  return 0;
}

void list_print_segment(list* l, int offset, int count){
  printf("L[%d:%d]: ... <->", offset, count);
  node* n = get_node(l, offset);
  if (count >= 0) for(int i = 0; i < count; ++i){
    printf("%d <-> ", n->value);
    n = n->next;
  }
  else for(int i = 0; i > count; --i){
    printf("%d <-> ", n->value);
    n = n->prev;
  }
  printf("...\n");
}
```

## B. Reporte Escrito

El reporte de la práctica deberá ser entregada en un archivo en formato `PDF` siguiendo las siguientes especificaciones:

- La primera página del documento deberá ser la carátula oficial para prácticas de laboratorio disponible en `lcp02.fi-b.unam.mx/`
- El nombre del documento `PDF` deberá ser `nn-XXXX-L08.pdf`, donde:
  - `nn` es el número de lista del alumno a dos dígitos forzosos (ej. 01, 02, etc.).
  - `XXXX` corresponden a las dos primeras letras del apellido paterno seguidas de la primera letra del apellido materno y la primera letra del nombre, en mayúsculas y evitando cacofonías; es decir, los cuatro primeros caracteres de su RFC o CURP.
- El reporte consiste en un documento de redacción propia donde el alumno explica de forma concisa y a detalle las actividades realizadas en la práctica, y reportando los resultados obtenidos.
- La longitud del reporte no deberá exceder las 3 páginas, sin contar la carátula.
- El reporte deberá seguir todos los lineamientos para documentos escritos establecidos al inicio del curso.
- Todas las referencias deberán estar debidamente citadas.

**IMPORTANTE:** No se aceptan archivos en otros formatos ni con nombres distintos a los especificados.