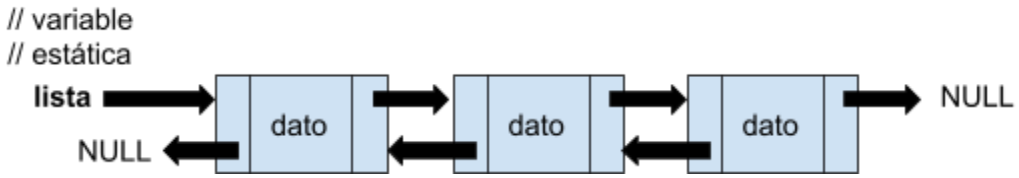
Listas doblemente vinculadas

Estructura de datos que consiste en un conjunto de nodos enlazados secuencialmente. Cada nodo contiene dos campos, llamados enlaces, que son referencias al nodo siguiente y al anterior en la secuencia de nodos. El enlace al nodo anterior del primer nodo y el enlace al nodo siguiente del último nodo, apuntan a un tipo de nodo que marca el final de la lista, normalmente un puntero null, para facilitar el recorrido de la lista.

El doble enlace de los nodos permite recorrer la lista en cualquier dirección. Mientras que agregar o eliminar un nodo en una lista doblemente enlazada requiere cambiar más enlaces que en estas mismas operaciones en una lista simplemente enlazada, las operaciones son más simples porque no hay necesidad de mantener guardado el nodo anterior durante el recorrido, ni necesidad de recorrer la lista para hallar el nodo anterior, la referencia al nodo que se quiere eliminar o insertar es lo único necesario.

Gráficamente, una lista doblemente enlazada (de cualquier tipo de dato) sería algo así:



Si se administra de una forma especial (con una estructura), el primer y el último nodo de una lista doblemente enlazada son accesibles inmediatamente (o sea, accesibles sin necesidad de recorrer la lista, y usualmente llamados cabeza y cola) y esto permite recorrerla desde el inicio o el final de la lista, respectivamente. Cualquier nodo de la lista doblemente enlazada, una vez obtenido, puede ser usado para empezar un nuevo recorrido de la lista, en cualquier dirección (hacia el principio o el final), desde el nodo dado.

Los enlaces que contiene un nodo de este tipo de lista son frecuentemente llamados anterior y siguiente o antecesor y sucesor.

La estructura de datos que se necesita para crear un nodo (y con ello la lista de nodos) es la siguiente:

```
typedef struct {
     tipo campo1;  // campos de datos, pueden ser estructuras
     tipo campo2;
     ........
     tipo campoN;
     struct nodoDoble * anterior; // dir de memoria nodo anterior
     struct nodoDoble * siguiente; // dir de memoria nodo siguiente
} nodoDoble ;
```

```
nodoDoble * listaDoble ; // variable estática y con nombre que
contiene la    // dirección de memoria del
// primer nodo de la lista. Si la lista está vacía, entonces

listaDoble = NULL;
```

Operaciones básicas del manejo de listas doblemente enlazada

Para el funcionamiento de esta colección de datos, debemos codificar las mismas funciones desarrolladas para la Lista Simple, adaptando el código al tratamiento de los dos enlaces que posee cada nodo, teniendo en cuenta las distintas posibilidades en cuanto a la posición del mismo (inicio de la lista, medio o final).

NOTA: Para ejemplificar la explicación, usamos la siguiente estructura de datos:

```
typedef struct {
     persona dato;
     struct nodoDoble * anterior;
     struct nodoDoble * siguiente;
} nodoDoble;
```

```
typedef struct {
     char nombre[20];
     int edad;
} persona;
```

```
nodoDoble * listaDoble; // variable estática definida en el main()
```

donde:

***dato** es una variable de tipo persona que contiene información.*

***anterior** es un campo puntero, que contiene la dirección de memoria de otra estructura similar, anterior a la actual. Si es el nodo inicial, esta apunta a NULL.*

***siguiente** es un campo puntero, que contiene la dirección de memoria de otra estructura similar, siguiente a la actual. Si es el nodo final, esta apunta a NULL.*

***listaDoble** es una variable estática (la declaramos a nivel de main) que contiene la dirección de memoria del primer nodo de la lista.*

A modo de ejemplo, a continuación se desarrollan algunas de las mencionadas funciones, es tarea del alumno completar el resto.

Inicializar la lista

Inicializa el puntero al primer nodo de la lista con el valor NULL.

```
nodoDoble * inicListaDoble() {
   return NULL;
}
```

Crear un nuevo nodo para luego agregar a la lista

Función que recibe como parámetro los campos de información para la estructura, los agrega a la misma y retorna un puntero al NODO creado.

```
nodoDoble * crearNodoDoble (persona dato) {

    nodoDoble* aux = (nodoDoble*) malloc(sizeof(nodoDoble));
    aux->dato = dato;
    //asigna valor NULL a los campos que contienen la dirección de memoria
    //de los nodos anterior y siguiente
    aux->anterior = NULL;
    aux->siguiente = NULL;

    return aux;
}
```

Agregar un nodo ya creado al principio de la lista

Función que recibe como parámetro una variable puntero al comienzo de la lista y otra variable puntero a un nuevo nodo. Agrega este nuevo nodo al comienzo de la lista y retorna la nueva posición de memoria.

```
nodoDoble * agregarPpioDoble (nodoDoble * lista, nodoDoble * nuevo) {
    nuevo->siguiente = lista;
    if(lista != NULL)
       lista->anterior=nuevo;
    return nuevo;
}
```

Buscar el último nodo (recursiva)

Función que nos retorna la dirección de memoria del último nodo de la lista. Se comparte este ejemplo recursivo, pensar cómo hacerlo de forma iterativa (es similar a la lista simple)

```
nodoDoble * buscarUltimoDobleRecursivo (nodoDoble  * lista) {
   nodoDoble * rta;
   if (lista==NULL)
      rta=NULL;
   else
      if(lista->siguiente==NULL)
         rta=lista;
      else
         rta=buscarUltimoDobleRecursivo(lista->siguiente);
   return rta;
 }
```

Agregar un nuevo nodo al final de la lista

Esta función nos permite agregar al final de la lista un nuevo nodo.

```
nodoDoble * agregarFinalDoble(nodoDoble  * lista, nodoDoble * nuevo) {

   if(lista == NULL) {
      lista = nuevo;
   } else {
      nodoDoble * ultimo = buscarUltimo(lista);
      ultimo->siguiente = nuevo;
      nuevo->ante = ultimo;
   }
   return lista;
}
```

Agregar un nodo nuevo manteniendo el orden (según un campo)

Esta función agrega un nodo nuevo a la lista, insertándolo en el lugar correspondiente según un orden preestablecido por un campo. Por ejemplo: insertar un nuevo nodo en la lista ordenada por el campo nombre. Retorna un puntero al primer nodo de la lista, que se modifica solamente cuando el nuevo nodo se inserta en la primera posición.

```
nodoDoble * agregarEnOrdenDoble (nodoDoble  * lista, nodoDoble * nuevo) {

   if(lista == NULL) {
      lista = nuevo;
   }else {

      if(strcmp(nuevo->dato.nombre,lista->dato.nombre)<0){
         lista = agregarPpioDoble(lista, nuevo);
```

```
        } else {
           // se puede recorrer la lista utilizando un solo puntero??
           nodoDoble * ante = lista;
           nodoDoble * seg = lista->siguiente;
           while((seg != NULL)
                    &&(strcmp(nuevo->dato.nombre,seg->dato.nombre)>0)) {
              ante = seg;
              seg = seg->siguiente;
           }
           ante->siguiente = nuevo;
           nuevo->anterior = ante;
           nuevo->siguiente = seg;
           if (seg!=NULL)
               seg->ante=nuevo;
        }
     }
     return lista;
 }
```

Las siguientes funciones, completan el TDA Lista doblemente vinculada, quedan como ejercicio para el alumno:

- Borrar un nodo de la lista buscándolo por el valor de un campo (se puede recorre la lista utilizando un solo puntero)
- Buscar un nodo
- Eliminar el primer nodo de una lista
- Eliminar el último nodo de una lista
- Mostrar un nodo
- Recorrer y mostrar la lista (usar la función anterior)

Subprogramas:

- Menú del sistema
- Cargar lista al principio
- Cargar lista al final
- Cargar lista en orden