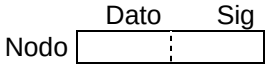
Listas

Una lista es un conjunto de datos del mismo tipo en el cual existe una relación de orden entre sus elementos (nodos). Los datos en las listas pueden ser contiguos o no.

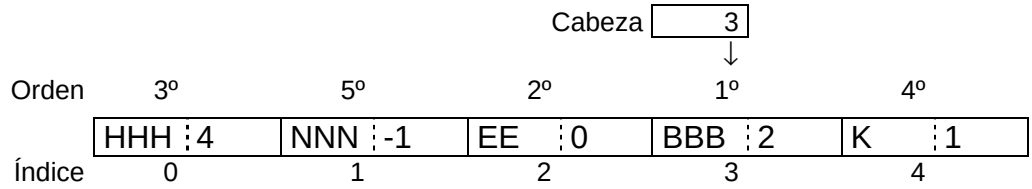
- Si son ***contiguos*** sus elementos se encuentran almacenados en forma adyacente y las operaciones de inserción y eliminación generan corrimientos (arreglo).

Orden	1º	2º	3º	4º	5º
	BBB	EE	HHH	K	NNN
Índice	1	2	3	4	5

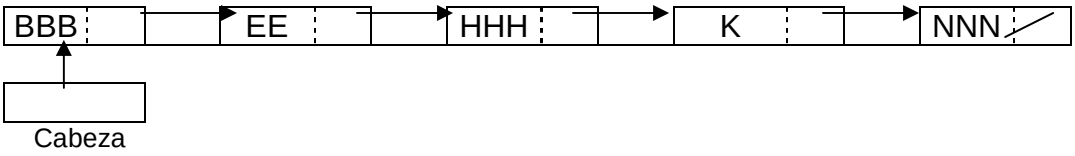
- Si ***no*** son contiguos se establece una relación de orden independientemente de la posición física de sus elementos. Cada nodo está formado por al menos dos campos, uno contiene el dato y otro la ubicación del siguiente elemento (creando un "enlace" al siguiente nodo). El último nodo no tiene siguiente, esto se indica con un valor especial en el campo siguiente (-1 o NULL).
  Las operaciones de inserción y eliminación implican modificar dichos enlaces, sin corrimientos. Debe especificarse cual es la ubicación del primer elemento (nodo) de la lista (cabeza).



Enlace Memoria Estática

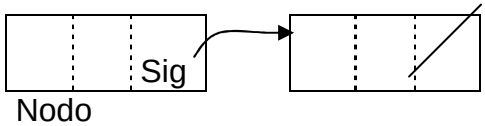


Enlace Memoria Dinámico



Enlace en memoria dinámica

A partir de " nodos" dinámicos, implementados como variables de tipo registro, donde un campo de tipo puntero a registro (Sig) contiene la dirección del próximo nodo, es posible implementar Listas, Pilas y Colas, tomando como enlaces los vínculos en memoria dinámica (a diferencia de los enlaces traves de arreglos de registros o cursores, en memoria estática).



la barra en un campo de tipo puntero indica el valor nil

Cada inserción implica crear y vincular un nodo dinámico y cada eliminación desvincularlo (devolviendo al heap los bytes que ocupa). La forma en que se modifican los enlaces o vínculos depende de la forma de operar de la estructura. No se considera la posibilidad de estructura Llena, ya que esta crece y decrece de acuerdo al requerimiento de almacenamiento.

```
typedef  int TELEMENTOL;
typedef  struct nodo{
   TELEMENTOL dato ;
   struct nodo   *sig;}NODO;
typedef  NODO  *TLISTA;
```

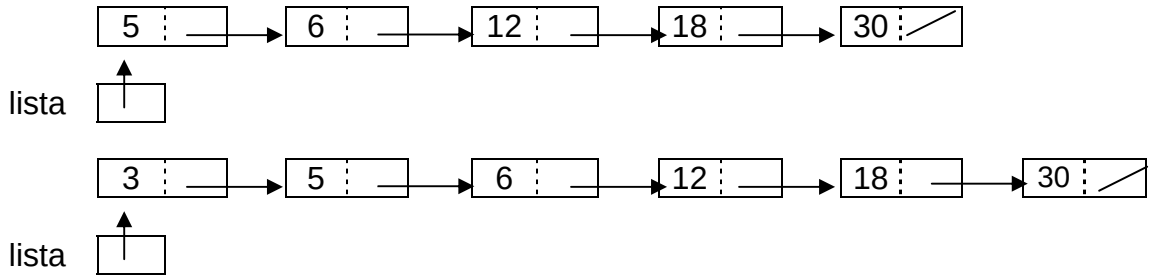
Iniciar una lista es prepararla para operar, debe estar vacía al comienzo

```
void inicial (TLISTA *pl)
 {    *pl= NULL;}
```

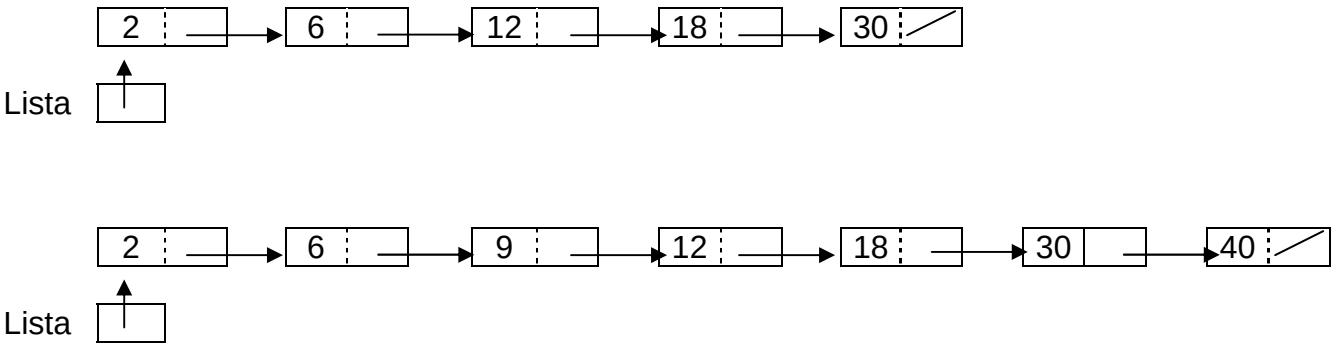
Una lista **vacía** cumple lista== NULL

Para **insertar** un elemento en una lista ordenada, se consideran dos casos:

1º_ Cuando la lista está vacía o el elemento a insertar es menor que el primero inserta al principio. Se modifica la cabeza de la lista. Ejemplo: se quiere insertar en una lista de números el 3



2º_ Cuando no va al comienzo, puede ser intermedio o último, se debe buscar el nodo anterior. Ejemplo 9 y 40




```
void insertal (TLISTA *pl,  TELEMENTOL elem)
{
  TLISTA   nuevo,  ant,  aux ;
   nuevo= (TLISTA)malloc(sizeof(NODO));
   nuevo->dato=elem;
  if   (*pl == NULL  ||  (*pl)->dato > elem)  /*inserta principio*/
       {
        nuevo->sig = *pl;   *pl= nuevo;
        }
   else
        {
        aux= *pl;
        while   (aux != NULL &&  elem > aux->dato)   /*inserta en el medio o al final*/
```

Nuevo dirección del nodo que ingresa
Ant dirección del nodo que le antecede
Aux dirección del nodo que le sigue

```
        {
         ant = aux   ;  aux = aux->sig;
         }
        ant->sig =  nuevo;
    nuevo->sig := aux;
    }
}
```

Para **eliminar** el n-ésimo elemento de la lista, se debe contar los nodos mientras se recorre la lista, se evalúan dos casos: es el primer nodo o es intermedio (o último).

```
void  eliminaln (TLISTA *pl, int n,  TELEMENTOL *pelem)
{
  int cont;   TLISTA  aux,  ant;
  if (*pl  != NULL)
      if  (n== 1)
          {
          *pelem = (*pl)->dato;
          aux =*pl;
          *pl=(*pl)->sig;
          free( aux);
          }
      else
          {
          cont= 1;   aux = *pl;
          while  (aux != NULL  &&  cont < n)
              {
              cont++;   ant = aux;  aux = aux->sig;
              }
          if (aux != NULL)
              {
              *pelem= aux->dato;
              ant->sig = aux->sig;          /*modifica el enlace anterior*/
              free( aux);
              }
          }
   }
```

aux dirección del nodo que sale
ant dirección del nodo anterior al que sale

Ejercicios utilizando Lista dinámica

Ej 1 -Recorrer una lista que contiene cadenas, mostrando las mismas e imprimir por fin de proceso la cantidad de cadenas con longitud mayor a 5.

```
#include <stdio.h>
#include <string.h>
#define  MAX    20
typedef  struct nodo{
   char cad[MAX] ;
   struct nodo   *sig;}NODO;
typedef  NODO  *TLISTA;

void cargalista(TLISTA *pl);
int main (void) {
     TLISTA  lista, aux;
     int cont =0 ;
     cargalista(&lista);
     aux= lista;
     while (aux != NULL)
          {
          if (strlen(aux->cad) > 5 )
               cont ++;
          puts(aux->cad);
          aux= aux->sig
          }
     printf ("la cantidad de cadenas de mas de 5 caracteres es %d ', cont);
     return 0;
     }
/*inserta por la cabeza, orden inverso al ingreso*/
void cargalista(TLISTA *pl){
TLISTA  aux;
char cadena[MAX] ;
*pl=NULL;
while(gets(cadena)!=NULL) /* ctrl-Z */
     {
     aux= (TLISTA)malloc(sizeof(NODO));
     strcpy(aux->cad, cadena);
     aux->sig=*pl;
     *pl=aux;
     }
}
```

```
/*inserta al final, respeta el orden de ingreso*/
void cargalista(TLISTA *pl){
TLISTA  aux;
char cadena[MAX] ;
*pl= (TLISTA)malloc(sizeof(NODO));
gets((*pl)->cad);
(*pl)->sig=NULL; aux=(*pl);
while(gets(cadena)!=NULL)
     {
     aux->sig= (TLISTA)malloc(sizeof(NODO));
     aux=aux->sig;
     strcpy(aux->cad, cadena);
     aux->sig=NULL;
     }
}
```

Ej 2- Sea una pila de palabras, se pide vaciarla y mostrarlas ordenadas alfabeticamente junto a su frecuencia.

```
#include <stdio.h>
#include <string.h>
#include "tdapila.h"
#define  MAX    20
typedef  struct nodo{
   char cad[MAX] ;
   int frecuencia;
   struct nodo   *sig;}NODO;
```

```
typedef  NODO  *TLISTA;

void cargapila(TPILA *ppila);
void calcula(TLISTA *plista);
void muestra(TLISTA lista);

int main (void) {
     TLISTA  lista, ant, aux;
     TPILA pila;
     int cont =0 ;
     char cadena[20];
     cargapila(&pila);
     lista=NULL;
     while(! vaciap(pila))
          {
          sacap(&pila, cadena);
          calcula(&lista, cadena);
          }
     muestra(lista);
     return 0;
}
void cargapila(TPILA *ppila){
     char cadena[20];
     iniciap(ppila);
     while(gets(cadena)!=NULL)
          ponep(ppila, cadena);
}

void calcula(TLISTA *plista, char cadena[]){
     TLISTA  nuevo, aux=*plista, ant=NULL;
     while(aux != NULL && strcmp(cadena, aux->cad)>0)
          {
          ant=aux;    aux=aux->sig
          }
     if  (aux != NULL && !strcmp(cadena, aux->cad))
               aux->frecuencia++;
     else
               {
                nuevo= (TLISTA)malloc(sizeof(NODO));
                strcpy(nuevo->palabra, cadena);
                nuevo->frecuencia= 1;
                nuevo->sig= aux;
                if (ant == NULL)
                     *plista= Nuevo;
               else
                     ant->sig= nuevo;
               }
}
void muestra(TLISTA lista){
     while ( lista !=NULL){
          printf("%s    %d  \n", lista->cad, lista->frecuencia);
          lista=lista->sig;}
}
```

Ej 3-Dada una lista que contiene letras, eliminar las vocales mayúscula y modificar las minúscula reemplazándolas por el siguiente caracter. Informar el porcentaje de vocales que contenía la lista.

```
#include <stdio.h>
#include <string.h>
typedef  struct nodo{
   char letra;
   struct nodo   *sig;}NODO;
typedef  NODO  *TLISTA;

void cargalista(TLISTA *pl);
int vocal(char c);
int main (void) {
     TLISTA  lista, aux, ant;
     int contc =0, contv=0 ;
     cargalista(&lista);
     aux= lista;   ant= NULL;
     while (aux !=NULL){
         if  (vocal(aux->letra))
              { contv++;
               if (islower(aux->letra   ))
                   {    aux->letra++;  aux= aux->sig;   }
              else   /*es mayuscula la elimino*/
                        if (ant == NULL)
                            { lista= lista->sig;  free(aux); aux=lista;    }
                        else
                            { ant->sig= aux->sig;    free(aux);   aux= ant->sig;   }
                }
         else
              {contc ++;  aux= aux->sig;    }
  printf("Porcentaje de vocales %f  \n", (float)contv * 100/(contv + contc));
  return 0;
}
void cargalista(TLISTA *pl){
TLISTA  aux;
char c;
*pl=NULL;
while(c=getchar()!=EOF) /* ctrl-Z */
     {
     aux= (TLISTA)malloc(sizeof(NODO));
     aux->letra=c;
     aux->sig=*pl;
     *pl=aux;
     }
}

int vocal(char c){
     c=tolower(c);
     return  c=='a' || c=='e' || c=='i' || c=='o' || c=='u';
     }
```