Listas simplemente enlazadas

Concepto

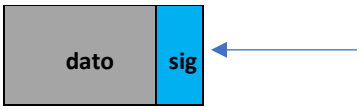
Una lista es una secuencia de 0 o más nodos de un mismo tipo almacenados en memoria. Son estructuras lineales: cada nodo (excepto el último) tiene sucesor y tiene anterior (excepto el primero). Una lista se puede implementar de forma tal que los nodos no estén físicamente contiguos, aunque se mantenga la relación de sucesión. Una implementación de este tipo es la de **lista enlazada**.

Una lista es una estructura dinámica ya que no es necesario reservar previamente memoria como en los arreglos o las matrices. Durante la ejecución del código, se reserva memoria dinámica para cada nodo de una lista, y se libera cuando ya no se necesita.

En una lista enlazada, a diferencia de una lista contigua, cada elemento sabe quién le sigue. Podemos hacer la analogía con una sala de espera de un médico: cada persona sabe quién le sigue, aun cuando el siguiente no se haya sentado al lado.

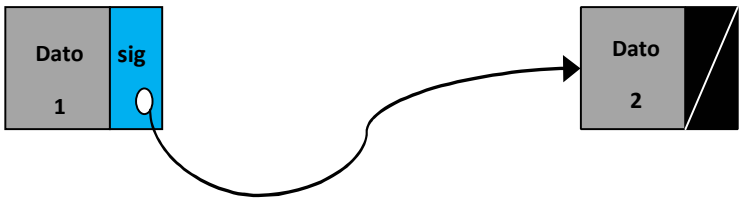
Entonces, en la lista enlazada, cada nodo tendrá la información propia y un enlace al siguiente elemento.

El enlace al siguiente nodo será a través de un puntero.



Si no tiene siguiente, el campo sig deberá estar en NULL.

Caso contrario, apuntará al siguiente:



La estructura de cada nodo en C la definiremos con Nodo

```
typedef  struct  Lista{
        int  clave ;
        struct  Lista *sig;
} Nodo  ;
```

Habrá que poner acá el tipo de dato que compone la lista. Podríamos tener un int, un float, un char, un arreglo o una estructura

Si en la lista no tenemos ningún elemento (lista vacía) la lista estará en NULL.

Operaciones sobre las listas enlazadas

1. Crear una lista.
2. Mostrar la lista.
3. Insertar elemento en la lista.
4. Eliminar un elemento de la lista.

En este módulo vamos a detallar cómo se crea y se muestra una lista.

Crear una lista simplemente enlazada

Para crear una lista debemos seguir lo siguientes pasos:

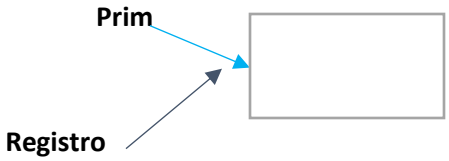
- Asignar memoria para un elemento del tipo de dato definido en la estructura y generar los siguientes nodos hasta que no haya más entrada de datos.

En el código:

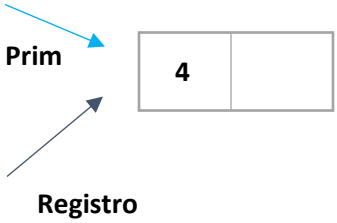
prin=(nodo *)malloc(sizeof(nodo));

En esta línea de código, reservamos memoria. La cantidad la indica sizeof y depende de la estructura definida. La dirección que devuelve malloc la recibe el puntero Prim.

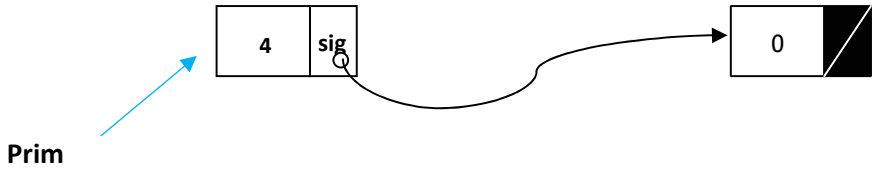
- Luego, llamamos a la función crear y le pasamos por parámetro el puntero Registro, esto es: Registro recibe la dirección donde apunta Prim.



- Le pedimos el dato al usuario y lo ingresamos a la lista.



- Hasta que sea cero el dato ingresado le pedimos más datos.



El código de la función Crear y el main es:

```
void crear (nodo *registro)
{

    registro->num =;

    if (registro->num==0)
        registro->sig=NULL;
    else
        {

        registro->sig=(nodo*)malloc(sizeof(nodo));

        crear (registro->sig);

        }
return;
}

int main()
{
    int i=0;
    nodo *prin;

    prin=(nodo*)malloc(sizeof(nodo));
    crear(prin);
    mostrar (prin);
    return 0;
}
```

MUY IMPORTANTE: ¡NO PERDER LA CABEZA!

Se usó un puntero auxiliar anterior y no se recorrió con el mismo puntero la lista para no perder su comienzo. Al puntero en el inicio de la lista se lo suele denominar “cabeza”, en este caso, Prim. Al efectuar ciertos recorridos sobre las listas hay que tener cuidado de “no perder la cabeza” y, por eso, es conveniente usar punteros auxiliares (Registro).

Mostrar una Lista

Ya creamos la lista cuyo puntero en la cabeza de lista es Prim. Llamamos a la función Mostrar que tiene como único parámetro este puntero. En dicha función, la dirección de Prim recibe Registro y empieza a recorrer la lista recursivamente, mostrando el elemento en pantalla.

El código de la función mostrar es:

```
void mostrar (nodo *registro)
{
    if (registro->sig !=NULL)
        {
        printf ("%d\n",registro->num);
        mostrar (registro->sig);
        }
        return;
}
```