**Curso:** ESTRUCTURAS DE DATOS
**Tema:** Estructuras de Datos
**Subtema:** Estructuras de datos Dinámicas - Listas Enlazadas Simples
**Profesor: Braulio Barrios Z. – Valledupar – Cesar - Colombia**

**¿Por qué son útiles las estructuras de datos?.** Cual es la importancia de que quienes se forman como futuros Ingenieros de Sistemas conozcan muy bien todo lo relacionado con las **Estructuras de Datos**?. La respuesta a estas preguntas está relacionada directamente con el papel que desempeña el Ingeniero de Sistemas en el diseño de nuevos sistemas de información. Una de las fases cruciales en la concepción de un sistema de información es la fase de DISEÑO, y para enfrentar las complejidades habituales del diseño los responsables de esta fase hacen uso de alguna forma de ABSTRACCION y en consecuencia, de alguna organización (estructura) de los datos.

**¿En que consiste la Abstracción?.** La complejidad del mundo que nos rodea sería un factor infranqueable para la mente humana si no hiciéramos habitualmente abstracciones de los distintos fenómenos de esta realidad que nos circunda. Así, por ejemplo, Un plano hecho por un arquitecto describe más o menos de manera completa la estructura de una casa que ha de construirse. De la misma forma que un mapa del Departamento del Cesar simplifica la adquisición de conocimiento por parte nuestra acerca de la realidad geográfica, física, hídrica, económica, etc, del departamento en el cual vivimos.

La abstracción es un proceso "***mental***" mediante el cual pasamos por alto ciertas características de un fenómeno y nos centramos en el detalle de aquellas que nos parecen relevantes, según el propósito que se persiga. Así las cosas, no necesitamos ver la casa terminada para tener una apreciación de ella; de la misma manera que no es necesario abordar un automóvil en Valledupar y dirigirnos al sur del Departamento para conocer como limitan los distintos municipios que conforman el Departamento.

En el desarrollo de Software, la abstracción es la clave para el logro de un buen producto final, toda vez que el desarrollo de un sistema informático se ve simplificado en gran medida mediante el uso de la abstracción en la fase de Diseño. Se trata entonces, en principio, de especificar en el Diseño la funcionalidad del sistema en términos generales lo cual se conoce también como "Especificaciones de alto nivel". Una vez que se demuestre que esta abstracción del sistema es correcta, entonces se le pueden añadir mas detalles; finalmente, se hace una especificación detallada (llamada también de bajo nivel) del sistema que estamos diseñando y es esta ultima especificación la que podemos expresar utilizando la sintaxis de un lenguaje de programación dado. En este nivel suele utilizarse un tipo especial de abstracción llamado **Abstracción de Datos**, la cual involucra dos fases, a saber: Descripción abstracta (lógica) de los datos que se requieren, y también de las operaciones que pueden realizarse con ellos.

**¿Qué es un Tipo Abstracto de Datos ( TAD ) ?** Es un modelo matemático de los objetos de datos que constituyen un tipo de datos, así como de las funciones que operan sobre estos objetos. Así, por ejemplo, el **TAD** Matriz puede utilizarse para representar matrices matemáticas junto con las operaciones definidas sobre matrices. Igualmente, el **TAD Vector_de_Registros_Alumnos** puede utilizarse para representar un Arreglo unidimensional en el cual cada elemento consta de un **Registro** de información del alumno (Código, Nombre, Calificación), conjuntamente con las siguientes operaciones definidas sobre estos datos:

**Registrar_Alumno(**Cod**)** : Inserta un registro de alumno en el vector
**Consultar_Alumno(**Cod**)** : Muestra la información de un alumno
**Corregir_nota(**Cod**)** : Actualiza la información de calificación del alumno
**Informe_de_notas(**Cod**)** : Muestra un listado de alumnos con todos los campos de cada registro.

Para concluir, diremos que un **TAD** es la suma de Las **ESTRUCTURAS DE DATOS** mas los **PROCEDIMIENTOS** o **FUNCIONES** que manipulan esos datos.

**¿Qué es una ESTRUCTURA DE DATOS?** Es una colección de variables en memoria que están relacionadas de alguna forma específica con el fin de facilitar el almacenamiento y acceso a la información.

**¿Cómo se Clasifican las ESTRUCTURA DE DATOS?**

Según el medio de almacenamiento, las estructuras de datos se clasifican en:

- ESTRUCTURAS EN MEMORIA PRINCIPAL (RAM)
- ESTRUCTURAS EN MEMORIA SECUNDARIA

- En este curso estudiaremos únicamente las estructuras de datos EN MEMORIA PRINCIPAL

**¿Cómo se Clasifican las ESTRUCTURA DE DATOS EN MEORIA PRINCIPAL?**

Se clasifican en:

| | | |
|---|---|---|
| **ESTRUCTURAS DE DATOS ESTATICAS** | | **ARREGLOS UNIDIMENSIONALES** (Vectores)<br>**ARREGLOS MULTIDIMENSIONALES** (Matrices) |
| **ESTRUCTURAS DE DATOS DINAMICAS** | **LINEALES** | **LISTAS ENLAZADAS SIMPLES**<br>**LISTAS ENLAZADAS DOBLES**<br>**LISTAS ENLAZADAS CIRCULARES**<br>**PILAS**<br>**COLAS** |
| | **NO LINEALES** | **ÁRBOLES**<br>**GRAFOS** |

Una ***ESTRUCTURA DE DATOS ESTATICA*** es aquella cuyo tamaño, en tiempo de Ejecución del programa **NO** puede alterarse una vez que se ha establecido; de ahí el calificativo de "**Estática**". Las estructuras de datos estáticas se identifican por loas siguientes características:

**1.**- DEBEN SER DIMENSIONADAS PREVIAMENTE (Definir la cantidad máxima de datos a almacenar)
**2**.- TODOS LOS ELEMENTOS EN LA ESTRUCTURA DEBEN SER DE IGUAL TIPO (Tipos de datos)
**3**.- LOS ELEMENTOS DE LA ESTRUCTURA SE ALMACENAN EN POSICIONES CONSECUTIVAS DE RAM.

Una ***ESTRUCTURA DE DATOS DINAMICA*** es aquella cuyo tamaño, en tiempo de Ejecución del programa **SI** puede alterarse, es decir, "crecen" o se "contraen" según la intervención del usuario de la misma.

Para abordar el estudio de las ESTRUCTURAS DE DATOS DINAMICAS necesitamos entender previamente el concepto de **APUNTADOR**.

Un **APUNTADOR es una variable especial, en el sentido de que su contenido NO es un dato de los tipos primitivos (Carácter, cadena, entero corto, entero largo, decimal, etc. ) sino una DIRECCION DE MEMORIA RAM, dato este expresado en sistema de numeración Hexadecimal.**

Hay que anotar que los Apuntadores surgen vinculados directamente a la necesidad de manipular "Dinámicamente" la memoria RAM. No todos los lenguajes de programación soportan (implementan) el uso de apuntadores; por ejemplo: BASIC, COBOL, FORTRAN no implementan este tipo de variables, mientras que PASCAL, ADA, MODULA2, C/C++, JAVA, DELPHI sí lo implementan y, en consecuencia, proveen las funciones (o procedimientos) para gestión dinámica de memoria mediante apuntadores.

En esta (y en las demás lecciones del curso) utilizaremos en los algoritmos el nombre de dos funciones de lenguaje PASCAL para gestión de apuntadores, y son las siguientes:

**NEW** (Apuntador) : Pide al sistema operativo memoria para ser asignada al apuntador

**DISPOSE** (Apuntador): Ordena al sistema operativo que disponga del bloque de memoria apuntado.

## LISTA ENLAZADA SIMPLE:

**Definición:** Una Lista Enlazada Simple. es una Estructura de datos dinámica que consiste en un Conjunto finito de elementos llamados **NODOS** de la estructura, tal que:

- **Por tratarse de un conjunto, si no existen nodos diremos que la lista es "Vacía".**

**O**

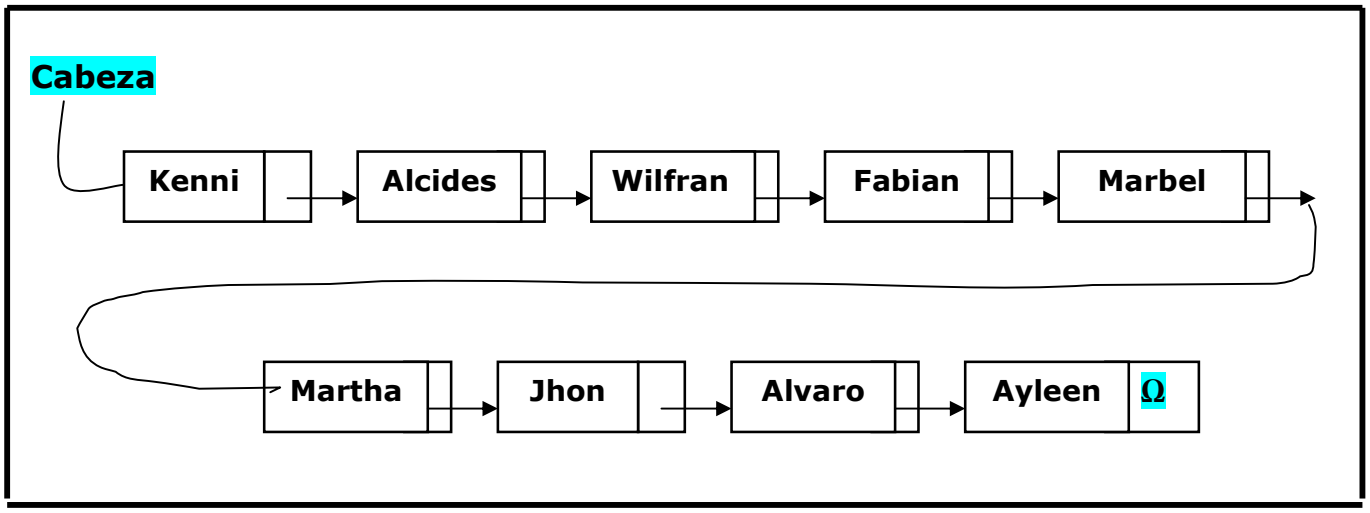
- **Si la lista NO es Vacía entonces cada NODO de la estructura tiene la siguiente composición:**


**INFO : Dato que se almacena en el Nodo**
**SIG : Apuntador que contiene la "Dirección" de memoria en la cual se encuentra almacenado el siguiente nodo de la lista (El byte de inicio del bloque de memoria que ocupa el nodo siguiente)**

**Existe un APUNTADOR externo a la estructura, el cual contiene la dirección de memoria del Primer Nodo de la Lista. A este apuntador se le conoce como "CABEZA".**

**El último nodo de la lista es aquel que en su sección "Siguiente" NO almacena ninguna dirección sino que contiene un valor propio de apuntadores llamado NIL, ó NULL, ó NULO; esta es la forma de saber donde "Termina" la lista enlazada.**

Gráficamente, podemos concebir una lista enlazada conteniendo nombres de alumnos en memoria RAM, como se indica en la siguiente figura:




Ahora bien, **Que se necesita, desde el punto de vista ALGORITMICO, para crear esta estructura de datos en memoria RAM ????.**

Lo primero es decidir el **tipo** de los datos que almacena cada **nodo** de la lista (Cadena, Carácter, Numérico entero, Numérico decimal, etc). A esto se le llama **"Definir la estructura de información".**

Tenga en cuenta que cada nodo almacena por lo general más de un campo de información, es decir, que conjuntamente con el nombre del alumno puede estar la cédula, la asignatura que cursa, la dirección de correo, etc. Hagámoslo para la lista de la gráfica anterior y definamos el nodo con SINTAXIS de **Lenguaje C++** así:

```
/* Declaración de la estructura*/

Struct Nodo
{
        char    Nombre[70];
        Nodo    * SIG;
};

/* Ahora declaramos las variables de tipo apuntador a esta estructura */

Nodo  *p,  *q,  *r,  *cabeza;
```

**La anterior definición expresa que : C**ada nodo de la estructura consta de: **Nombre** (cadena de 70 caracteres) y un apuntador **SIG** (Abreviatura de Siguiente) que señala al siguiente nodo de la lista.

Si el lenguaje escogido para la implementación del programa es **PASCAL** entonces habrá que definir la **Estructura de Información** que soportará nuestra lista enlazada simple así:

```
PROGRAM  listas;
USES
        wincrt;

TYPE   N = ^NODO;
NODO      = RECORD
                NOM : STRING[60];
                SIG  : N;
              END;

VAR
   (* Declaro ahora mis Variables apuntadores *)
    CAB, P, Q, R, ULTIMO : N;
```

**Ahora sí:** Iniciemos el estudio en detalle de nuestra estructura de datos dinámica **LISTA ENLAZADA SIMPLE**, para lo cual, y aplicando los conceptos enunciados al inicio de esta lección, consideraremos un TIPO ABSTRACTO DE DATOS (TAD) "**PACIENTES**", consistente en:

**a)** Una Estructura de datos Lista Enlazada Simple, con información de: **IDENTIFICACION, NOMBRE, NOMBRE, DIAGNOSTICO, ESTRATO, TARIFA.**

**b)** Las siguientes operaciones sobre esa estructura:

**Registrar_Paciente** añade un nodo a la estructura con información de un paciente
**Consultar_Historia** Muestra en pantalla los datos de un paciente solicitado
**Actualiza_Informacion_Paciente** Permite modificar DIAGNOSTICO y/o ESTRATO
**Informe_de_Pacientes** Muestra en pantalla un Informe de pacientes
**Retirar_Historia_de_Paciente** Borra un registro de paciente de la lista.

La **TARIFA** se calcula según el **ESTRATO** así:

| ESTRATO | TARIFA |
|---|---|
| <= 2 | 1.000 |
| 3 | 1.500 |
| >= 4 | 10.000 |

Lo primero es **DECLARAR LA ESTRUCTURA DE INFORMACIÓN** en la cual se soportará nuestro **TAD,**para lo cual, arbitrariamente he decidido utilizar en este pseudocódigo sintaxis de lenguaje C++ así:

```
/* Declaración de la estructura*/

Struct   Nodo
{
        long    IDENT;
        char    NOMBRE[40];
        int     EST;
        float   TARIFA ;
        char    DIAG[120] ;
        Nodo   * SIG;
};
/* Ahora declaramos las variables de tipo apuntador a esta estructura */
Nodo  *p,  *q,  *r,  *cabeza;
```

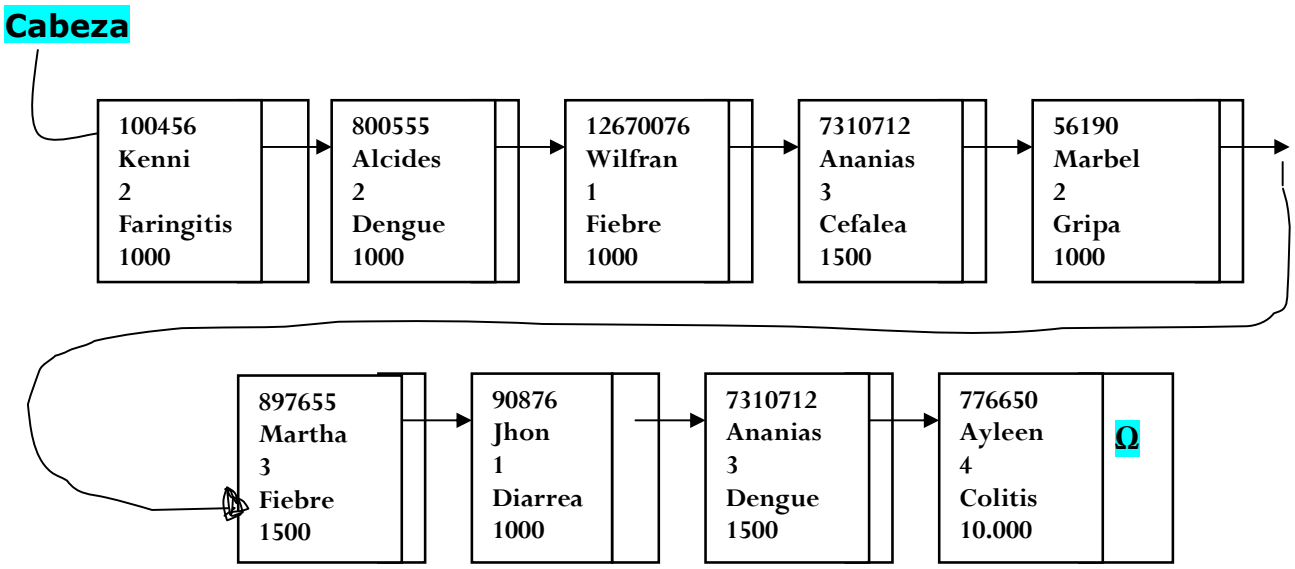
```
REGISTRAR_PACIENTE
        Seguir='s';
        MQ  Seguir ='s' Haga
          /* Pido un nodo */
         NEW(p);
         SI  p = NIL ent
                Escriba(" No hay memoria disponible en el sistema ....");
          SINO
              /* Llenamos el nodo ..*/
              Escriba("Introduzca Documento de Identidad      : "); Lea( p ^. IDENT )
              Escriba("Introduzca Nombre    : "); Lea(  p ^. NOMBRE )
              Escriba("Introduzca Estrato (Numérico 0- 6   : "); Lea( p ^. EST )
              Escriba("Introduzca Diagnóstico      : "); Lea( p ^. DIAG )

              /* Calculo de Tarifa*/
              SI  EST <= 2  Ent
                  p ^. TARIFA ← 1000
                 SINO
                     SI  EST = 3  Ent
                            p ^. TARIFA ← 1500
                       SINO
                            p ^. TARIFA ← 10000
                     FSI
              FSI

              /* Balanceamos el nodo ...*/
              P ^. SIG ← NIL

              /* es el primer nodo de la lista ???     */
              SI cabeza= NIL ent
                     Cabeza ← p
                SINO
                     /* recorremos la lista desde la cabeza hasta el ultimo nodo */
                     /* para añadir al final el nuevo alumno a  la lista */
```

```
                    r ← cab;
                    MQ  r  ^. SIG <> NIL Haga
                          r ← r  ^. SIG
                    FIN MQ
                    /* ahora el apuntador r apunta al ultimo nodo de la lista */
                    /* entonces añado el nuevo nodo después del ultimo */
                    r ^. SIG ← p
               FSI
          FSI
               Escriba (" Desea agregar otro registro de PACIENTE  a la lista? S/N "); Lea (seguir)
     FMQ
FIN REGISTRAR_PACIENTE
```

Observe que la condición es que **r ^. sig** sea diferente de **Ω, No que r sea diferente de Ω,** ya que si lo hiciéramos así entonces se **PASARÍA** del **ULTIMO** nodo de la lista .

==========

Con lo cual nuestra lista sería parecida a esta:




Observe que el paciente **Ananias** (7310712) tiene dos (**2**) registros en la lista

==========

**CONSULTAR_HISTORIA**

```
/* usted debe pedir al usuario la CEDULA del PACIENTE a consultar en la lista*/
/* luego recorre la lista con un apuntador desde la cabeza hasta el ultimo nodo*/
/* y va preguntando si la IDENT de cada nodo es igual a la que da el usuario para consulta*/
```

**FIN CONSULTAR_HISTORIA**

Veamos ahora el algoritmo en pseudocódigo para este proceso:

```
CONSULTAR_HISTORIA (Versión_1.0)
    Seguir ← 'S';
    MQ Seguir = ´S´Haga
        Escriba (" Identificación  a consultar? "); Lea(dato)
        P ← Cabeza; Encontro ← 'N' ; Cont ← 0
        MQ (Encontro='N')  AND (P  <> Ω )  Haga
            Cont ← Cont + 1
            SI   P ^. IDENT = dato Ent.
                 Encontro ← 'S'; Pos ← P
            SINO
                 P ← P ^. Sig ;
            FSI
        FMQ
        SI Encontro = 'S'  Ent
            Escriba("Encontrado el Paciente en la posición ", Cont)
        SINO
            Escriba("El Paciente consultado  NO está en la lista ……. ")
        FSI
        Escriba("Desea consultar Otro Paciente? [S/N]  "); Lea (Seguir)
    FMQ
FIN CONSULTAR_HISTORIA (Versión_1.0)
```

Observe que la condición es que **P** sea diferente de **Ω, No que P^.sig sea diferente de Ω,** ya que si lo hiciéramos así entonces no entraría al ULTIMO nodo de la lista y es posible que el dato buscado sea justamente el último

Observe la forma de "***moverse***" al siguiente Nodo. NO podemos decir que a **P ← P+1**, ya que NO se trata de una variable numérica sino de un **APUNTADOR**

Estudie el algoritmo anterior y hágale una **Prueba de Escritorio** para lo cual suponga que deseamos buscar La cédula **776650**; El algoritmo mostrará: " **Encontrado el Paciente en la posición : 9** ". Y si la cedula buscada es **554960** El algoritmo mostrará: "**El Paciente consultado NO está en la lista …….** ".

De igual manera, si buscamos la cédula **7310712** El algoritmo mostrará: "**Encontrado el Paciente en la posición : 4**". ( **¿**Puede usted explicar porqué esto ultimo**?**). Observe que la Cedula **7310712** está en **DOS** posiciones distintas dentro de la lista (Posición **4** y posición **8**), sin embargo, el algoritmo solo detecta la **PRIMERA OCURRENCIA** del elemento en la estructura de datos.

Si por exigencias del problema que estemos resolviendo necesitáramos detectar **TODAS** las ocurrencias del elemento entonces utilizaremos **OTRA VERSION** de este mismo algoritmo, como se muestra a continuación:

```
CONSULTAR_HISTORIA (Versión_2.0)
    Seguir ← 'S';
    MQ Seguir = ´S´Haga
        Escriba (" Identificación ? "); Lea(dato)
        P ← Cabeza; Encontro ← 'N' ; Cont ← 0;
        MQ ( P<> Ω ) Haga
            Cont ← Cont + 1
            SI   P ^.valor = dato Ent.
                 Encontro ← 'S' ; Pos ← P
                 Escriba("Encontrado el Paciente en la posición ", cont)
            SINO
                 P ← P ^. Sig ;
            FSI
        FMQ
        SI Encontro = 'N'  Ent
            Escriba("El valor consultado  NO está en la lista ……. ")
        FSI
        Escriba("Desea consultar Otro dato? [S/N]  "); Lea (Seguir)
    FMQ
CONSULTAR_HISTORIA (Versión_2.0)
```

Observe que la condición es que **P** sea diferente de **Ω, No que P^.sig sea diferente de Ω,** ya que si lo hiciéramos así entonces no entraría al ULTIMO nodo de la lista y es posible que el dato buscado sea justamente el último

Observe la forma de "***moverse***" al siguiente Nodo. NO podemos decir que a **P ← P+1**, ya que **NO** se trata de

**ACTUALIZA_INFORMACION_PACIENTE**

/* usted debe pedir al usuario la **CEDULA** del Alumno a modificar en la lista*/
/* luego recorre la lista con un apuntador desde la **cabeza** hasta el **ultimo** nodo*/
/* y va preguntando si la **IDENT** de cada nodo es igual a la que da el usuario para Modificar*/
/* si lo encuentra se detiene y pide las correcciones (Nuevo DIAGNOSTICO y/o Nuevo ESTRATO)
/* Si es necesario RECALCULA la **TARIFA** */

**FIN ACTUALIZA_INFORMACION_PACIENTE**

---

**RETIRAR_HISTORIA_DE_PACIENTE**

Es necesario indicar que "**RETIRAR**" información de una lista Enlazada implica "**BORRAR**" físicamente el nodo de la estructura de datos. Esto es porque como hemos dicho las listas enlazadas son Estructuras de datos **DINAMICAS**, es decir, que realmente se ***CONTRAEN*** cuando se les retira un elemento.

***Casos Especiales Al BORRAR un Nodo de una Lista Enlazada:***

**1.- BORRAR EL PRIMER NODO DE LA LISTA ENLAZADA**.
En este caso bastará con mover la cabeza al siguiente Nodo de la lista y luego Liberar el nodo.

**2.- BORRAR EL ÚLTIMO NODO DE LA LISTA ENLAZADA.**
En este caso bastará con colocarle **NIL** (**Ω**) a la sección SIGUIENTE del Penúltimo nodo de la lista y luego Liberar el nodo.

**3.- BORRAR UN NODO INTERMEDIO DE LA LISTA ENLAZADA**.
En este caso se requiere "Desenganchar" el nodo que contiene el elemento a borrar y restablecer correctamente los apuntadores para que no haya pérdida de información en la lista.

En cualquiera de los tres casos hay que verificar *PREVIAMENTE* si la LISTA **ESTA VACIA ó NO, para esto verificamos si CABEZA = Ω. (No olvide asignar Ω. Al apuntador Cabeza en el programa principal de su aplicación.**

**FIN RETIRAR_HISTORIA_DE_PACIENTE**

Entonces aprendamos como Borrar un nodo de nuestra Lista enlazada Simple:

**CONTINUA EN LA SIGUIENTE PAGINA ……**

```
RETIRAR HISTORIA DE PACIENTE
    Seguir ← 'S';
    MQ Seguir = ´S´ Haga
        Escriba (" Cedula del Paciente cuya historia va a Retirar ? "); Lea(dato)
        SI  Cabeza = Ω  Ent
            Escriba(" La lista está VACIA ")
        SINO
            P ← Cabeza; Encontro ← 'N' ;
            MQ (Encontro= 'N') AND  (P <> NIL) haga
                SI  P ^.valor  =  dato ENT
                    Encontro ←  'S'
                SINO
                    P ← P ^.Sig
                FSI
            FMQ
            SI Encontro = 'N'  ENT
                Escriba(" No está ese dato en la lista ….")
            SINO
                /* Está en el ULTIMO nodo???? */
                SI  P ^. Sig = Ω   ENT   /* Si, está de ultimo….*/
                    /*Entonces ubicamos OTRO apuntador ANTES de P */
                    Q ← Cabeza
                    MQ  Q ^. Sig < > P Haga
                        Q ← Q ^. Sig
                    FMQ
                    /* Ya Q está apuntando a un nodo ANTES de P, entonces
                        "Desengancho" el nodo al cual apunta P */
                    Q ^. Sig ← P ^. sig
                    /* Y ahora "Libero" el nodo al cual apunta P */
                    DELETE (P)
                SINO
                    /* Está en el PRIMER  nodo???? */
                    SI  P = Cabeza   ENT   /* Si, está en el PRIMERO ….*/
                        Cabeza ← Cabeza ^. Sig
                        /* Y ahora "Libero" el nodo al cual apunta P */
                        DELETE (P)
                    SINO
                        /* Está en un nodo INTERMEDIO cualquiera */
                        /*Entonces ubicamos OTRO apuntador ANTES de P */
                        Q ← Cabeza
                        MQ  Q ^.Sig < > P Haga
                            Q ← Q ^. Sig
                        FMQ
                        /* Ya Q está apuntando a un nodo ANTES de P, entonces
                            "Desengancho" el nodo al cual apunta P */
                            Q ^.Sig ← P ^. Sig
                        /* Y ahora "Libero" el nodo al cual apunta P */
                            DELETE (P)
                    FSI
                FSI
            FSI
        FSI
        Escriba(" Desea Borrar otro valor? [S/N] "); Lea(Seguir)
    FMQ

FIN RETIRAR HISTORIA DE PACIENTE
```

**En el algoritmo anterior, Observe usted que las secciones que están en colores ROJO y AZUL (**Que además están encerradas en recuadro) **son idénticas; Esto le sugiere a usted algo???. Será posible entonces "REESCRIBIR" este algoritmo y hacer una Versión 2.0 del mismo que aproveche esta circunstancia que indico para que así el algoritmo resulte mas sencillo, mas corto????.**