UASLP
Universidad Autónoma de San Luis Potosí

MANUAL DE PROGRAMADOR

Estructuras de Archivos

Héctor Andrey Hernández Alonso

FACULTAD DE INGENIERÍA

INDICE

ESTRUCTURA DEL PROYECTO ----------02

Interfaz Ventana Principal ----------03

Lectura y Escritura de Archivos Binario ----------04

Metadatos Entidad ----------05

Inserción Entidad ----------07

Eliminación Entidad ----------10

Modificación Entidad ----------12

Metadatos Atributo ----------17

Inserción Atributo ----------20

Eliminación Atributo ----------23

Modificación Atributo ----------27

Ejemplo Diccionario de Datos ----------29

Registros ----------30

Inserción Registro ----------32

Eliminación Registro ----------33

Modificación Registro ----------34

Tabla Registro ----------36

Índices ----------37

Reservación de espacio para Índices ----------38

Inserción Índice Primario ----------42

Inserción índice Secundario ----------43

Árbol B+ ----------45

Inserción árbol B+ ----------46

Eliminación árbol B+ ----------49

ESTRUCTURA DEL PROYECTO

Clases Principales

- Principal
- Entidad
- Atributo
- Registro
- Primario
- Secundario
- Arbol

Cada clase a su vez utliza otras clases para el manejo total de proyecto. En cada apartado se especificará que clases se usaran.

Caracteristicas del Proyecto

- Desarrollado en Visual Studio Community (2015)
- Lenguaje C#
- Tipo de Interfz: Windows Form

Referencias:

Estructuras de Datos

Osvaldo Cairó

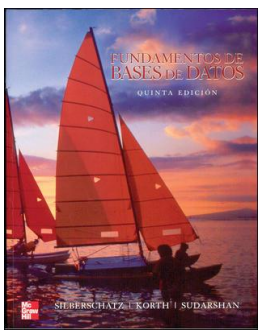
Tercera Edición

McGrawHill

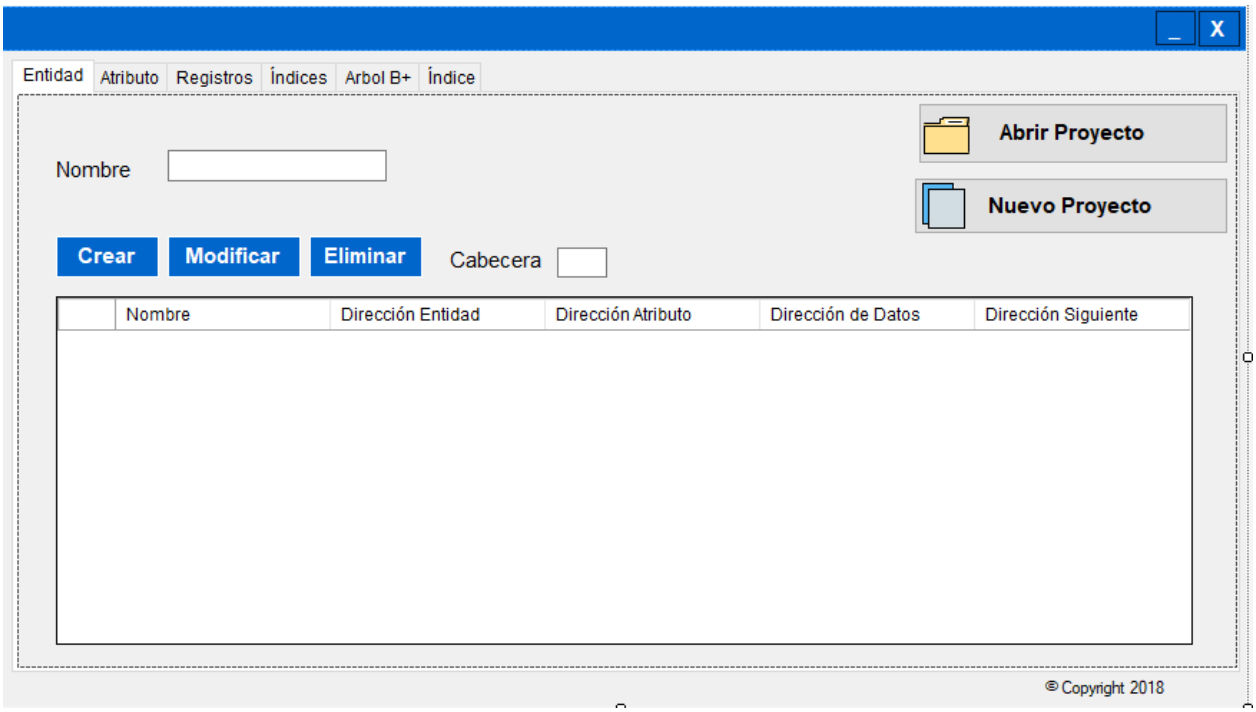
Fundamentos de Bases de Datos

Quinta Edición

McGrawHill



Interfaz Ventana Principal (Entidad)



Para poder controlar todo el proyecto, lo primero es contar con un Archivo que funcione como **diccionario de datos** para poder administrar todas las altas y bajas de Entidades y Atributos.

Diccionario de Datos

Es donde se almacenan las descripciones de los datos y sea accesible para los usuarios, contiene información que describe los datos de la base de datos (metadatos).

Normalmente un diccionario de datos describe entre otros casos:

- Nombre, tipo y tamaño de los datos (Entidades y Atributos).
- Relaciones entre datos.
- Restricciones de integridad sobre los datos.
- Usuarios autorizados a acceder a los objetos de base de datos.
- Estadísticas de utilización, tales como la frecuencia de las transacciones y el número de accesos realizados a los objetos de la base de datos.

Lectura y Escritura de Archivos Binarios

Para la lectura de archivos se necesita usar objetos de la Clase BinaryReader.

BinaryReader

Lee tipos de datos primitivos como valores binarios en una codificación específica.

```
BinaryReader br = new BinaryReader( File.Open( filename, FileMode.Open ) )
```

Filename → Nombre del Archivo

FileMode → Modo en que se manejara el archivo (Apertura, Escritura, Crear, etc.)

Referencia:

https://docs.microsoft.com/en-us/dotnet/api/system.io.binaryreader?view=netframework-4.7.2

BinaryWriter

Escribe tipos primitivos en binario en una secuencia y admite escribir cadenas en una codificación específica.

```
BinaryWriter bw = new BinaryWriter( File.Open( filename, FileMode.Open ) )
```

Referencia:

https://docs.microsoft.com/en-us/dotnet/api/system.io.binarywriter?view=netframework-4.7.2

Se considera que las Entidades y los Atributos están en un mismo archivo.

Los archivos son binarios.

Para la lectura de archivos binarios se usa un objeto de la clase BinaryReader

Entidades

Las entidades son los objetos reales que nos interesan lo suficiente como para capturar y guardar sus datos en una base de datos

- Metadatos de la Entidad
- Inserción Entidad
- Eliminación Entidad
- Modificación Entidad

METADATOS DE LA ENTIDAD

Nombre
Identificador propio del objeto que se requiere representar

Dirección de la entidad
Representa la dirección de archivo en donde se almacena la entidad

Dirección de los atributos
Representa la dirección de archivo en donde se encuentra el primer atributo.

Dirección de los datos
Representa la dirección de archivo en donde se encuentra el primer registro de datos.

Dirección de la siguiente entidad
Representa la dirección de la siguiente entidad

Nombre	DE	DA	DD	DSIG

NOMBRE	SIMBOLOGIA	METODO
Nombre	Nombre	br.ReadString()
Dirección Entidad	DE	br.ReadInt64()
Dirección Atributo	DA	br.ReadInt64()
Dirección Registro	DD	br.ReadInt64()
Dirección Siguiente	DSIG	br.ReadInt64()

Para poder controlar las Entidades, se utiliza una **Clase Entidad**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
/*********************************************************************************************
 * Clase Entidad
 * Constructor: Se inicializan los metadatos con valor   -1
 *
 *********************************************************************************************/

namespace Diccionario_de_Datos
{
    class Entidad
    {
        private string nombre;  // nombre Entidad
        private long DE;        // Dirección Entidad
        private long DA;        // Dirección Atributo
        private long DD;        // Dirección del Registro
        private long DSIG;      // Direccioón Siguiente Entidad
        private int i;

        // Constructor
        public Entidad()
        {
            nombre = "";
            DE = -1;
            DA = -1;
            DD = -1;
            DSIG  = -1;
            i = 0;

        }
```

La Clase Entidad cuenta solo con los métodos get's y set's de sus metadatos.

Los algoritmos que se presentan en el manejo de Entidades son:

- Inserción de Entidad
- Eliminación de Entidad
- Modificación de Entidad

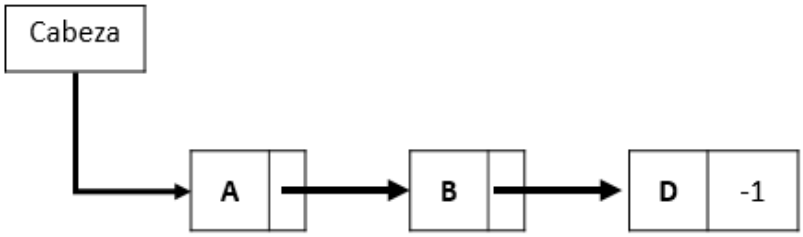
Algoritmo: Inserción de Entidad

Consideraciones al insertar:

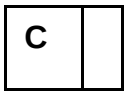
- Inserción Ordenada
- El primer campo de 8 bytes del archivo es la Cabeza de la lista, este indica la primera Entidad de forma ordenada
- La dirección de cada Entidad siempre será el tamaño del Archivo en ese momento

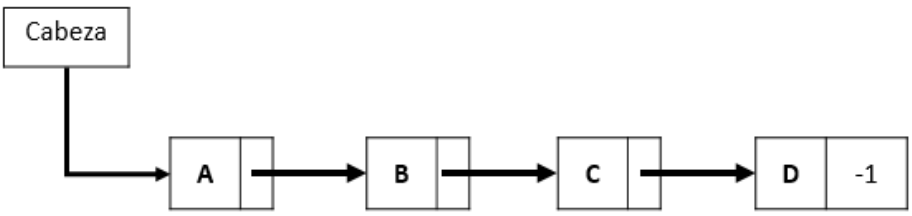
Pasos para la inserción de Entidades

1) Leer la Cabeza del Archivo y posicionar el apuntador. Si la Cabeza del Archivo indica -1, entonces no hay ninguna Entidad en el Archivo.
2) Buscar la nueva posición de la Entidad en el archivo.
3) Agregar la entidad con sus campos y actualizar su apuntador al siguiente
4) Si la entidad nueva, es la primera alfabéticamente, entonces la Cabeza del archivo apunta a la nueva Entidad



Insertar (C)





Para comparar las cadenas se usa el método CompareTo()

CompareTo()

Compara dos objetos y devuelve un valor entero con 3 posibles resultados donde:

- Si el valor es -1, La entidad 1 es antes que la nueva entidad
- Si el valor es 0, La entidad 1 y la nueva son las mismas
- Si el valor es 1, La entidad 1 va después de la nueva entidad.

```
int compara;

compara = entidad1.nombre.CompareTo(nuevaEntidad.nombre);
```

Referencia

https://docs.microsoft.com/en-us/dotnet/api/system.int32.compareto?view=netframework-4.7.2

```
269                 enti.nombrate(_nombre);
270                 enti.direccionate(bw.BaseStream.Length);
271                 enti.ponteDireccionAtributo(-1);
272                 enti.ponteDireccionRegistro(-1);
273                 dir = buscaEntidad(enti.dameNombre()); // Se obtiene la dirección de la entidad siguiente
274                 enti.ponteDireccionSig(dir);
275
276                 if (posicion == 1) // Actualización de Cabecera
277                 {
278                     bw.Seek(0, SeekOrigin.Begin);
279                     cabecera.Text = enti.dameDE().ToString();
280                     Cab = enti.dameDE();
281                     bw.Write(Cab);
282                 }
283                 bw.Seek((int)bw.BaseStream.Length, SeekOrigin.Begin); //Posiciona al final del Archivo
284                 bw.Write(enti.dameNombre());
285                 bw.Write(enti.dameDE());
286                 bw.Write(enti.dameDA());
287                 bw.Write(enti.dameDD());
288                 bw.Write(enti.dameDSIG());
289             }
290             entidad.Add(enti);
```

```
232     private void creaEntidad(object sender, EventArgs e)
233     {
234        enti = new Entidad();
235        List<string> nombresOrdenados = new List<string>();
236        long Cab = -1;
237        long dir = -1;
238        pSig = -1;
239        int cont;
240        _nombre = textBox1.Text;
241          //  comboBox1.Items.Add(_nombre); //Agrega las entidades al comboBox de atributo
242
243        cont = _nombre.Length;
244        for (; cont < 29; cont++){
245            _nombre += " ";
246        }
247        cEntidadRegistro.Items.Add(_nombre);
248            //comboBox6.Items.Add(_nombre);
249        if (entidad.Count == 0)
250        {
251            enti.nombrate(_nombre);
252            enti.direccionate(bw.BaseStream.Length);
253            enti.ponteDireccionAtributo(-1);
254            enti.ponteDireccionRegistro(-1);
255            enti.ponteDireccionSig(-1);
256
257            bw.Seek(0, SeekOrigin.Begin);
258                bw.Write(enti.dameDE()); // Cabecera
259                cabecera.Text = enti.dameDE().ToString();
260                Cab = enti.dameDE();
261                bw.Write(enti.dameNombre());
262                bw.Write(enti.dameDE());
263                bw.Write(enti.dameDA());
264                bw.Write(enti.dameDD());
```

```
267            else
268            {
269                enti.nombrate(_nombre);
270                enti.direccionate(bw.BaseStream.Length);
271                enti.ponteDireccionAtributo(-1);
272                enti.ponteDireccionRegistro(-1);
273                dir = buscaEntidad(enti.dameNombre()); // Se obtiene la dirección de la entidad siguiente
274                enti.ponteDireccionSig(dir);
275                //MessageBox.Show("Valor de dir: " + dir);
276
277                if (posicion == 1) // Actualización de Cabecera
278                {
279                    bw.Seek(0, SeekOrigin.Begin);
280                    cabecera.Text = enti.dameDE().ToString();
281                    Cab = enti.dameDE();
282                    bw.Write(Cab);
283                }
284                //MessageBox.Show("pSig: " + pSig);
285                // entidad[entidad.Count - 1].ponteDireccionSig(pSig);
286                bw.Seek((int)bw.BaseStream.Length, SeekOrigin.Begin); //Posiciona al final del Archivo
287
288             //   bw.Write(enti.dameDE());
289                bw.Write(enti.dameNombre());
290                bw.Write(enti.dameDE());
291                bw.Write(enti.dameDA());
292                bw.Write(enti.dameDD());
293                bw.Write(enti.dameDSIG());
294            }
295            entidad.Add(enti);
296
297            imprimeLista(entidad);
298
299        }
```

Algoritmo: Eliminación de Entidad

Consideraciones al eliminar:

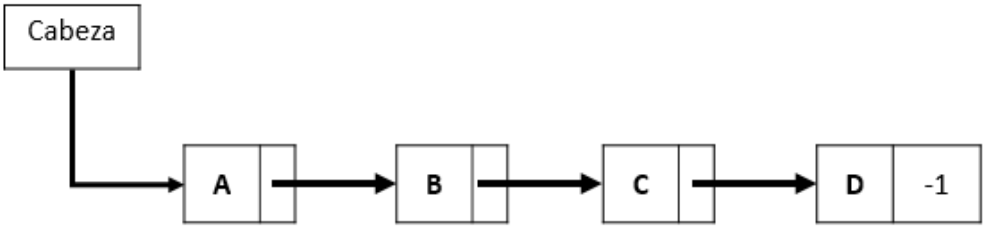
- Las Entidades no se eliminan, solo se deja de apuntar a ellas.
- Se debe guardar en un apuntador auxiliar el apuntador que apunta a la Entidad a Eliminar
- Se debe guardar en un apuntador auxiliar el apuntador siguiente de la Entidad a eliminar
- Si se elimina la primera entidad, actualizar la Cabeza de la lista con su siguiente

Pasos para la Eliminación de Entidad:

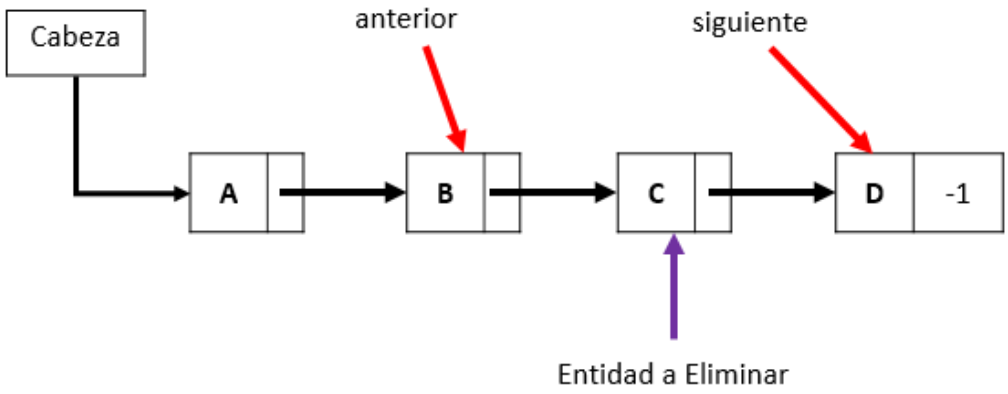
1) Leer la Cabeza del Archivo
2) Buscar en la lista la entidad que se desea eliminar
3) Obtener sus apuntadores anterior y siguiente
4) Enlazar su anterior ahora apunta a su siguiente

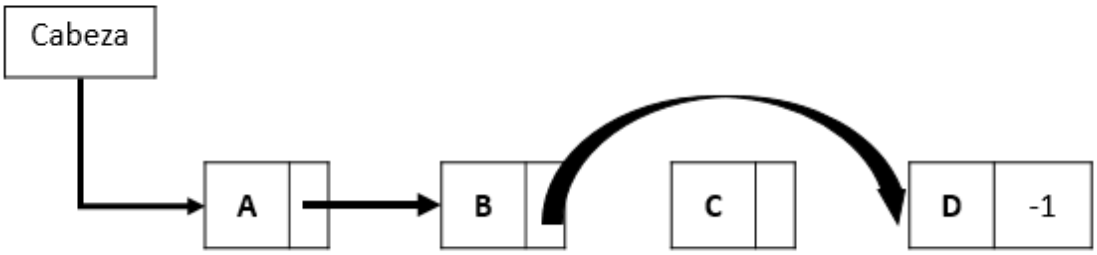
```
        private void eliminaEntidad(object sender, EventArgs e)
        {
            br.Close();
            bw.Close();
            string n = textBox1.Text;
            while (n.Length < 29)
            n += " ";
            long anterior;
            anterior = dameEntidadAnterior(n);
            long siguiente;
            string nn = " ";
            siguiente = dameSiguienteEntidad(n);
            br = new BinaryReader(File.Open(nArchivo,FileMode.Open));
            br.BaseStream.Position = br.ReadInt64();
            nn = br.ReadString();
            br.Close();
            bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
            if(nn == n)
            {
                bw.BaseStream.Position = 0;
                bw.Write(siguiente);
                cabecera.Text = siguiente.ToString();
            }
            bw.BaseStream.Position = anterior + 30 + 8 + 8 + 8;
            if(anterior == -1)
            {
                bw.BaseStream.Position = 0;
            }
            bw.Write(siguiente);
            bw.Close();
            imprimeLista(entidad);
        }
```

Ejemplo gráfico de Eliminación:



Eliminar (C)





Algoritmo: Modificación de Entidad

Una modificación es un poco más compleja ya que requiere de una eliminación como de una inserción por lo que se necesita guardar en auxiliares las direcciones a modificar.

Consideraciones al modificar:

- Apuntador que tenga la dirección anterior de la Entidad a modificar
- Apuntador que tenga la dirección siguiente de la Entidad a modificar
- Apuntador que tenga la nueva dirección anterior
- Apuntador que tenga la nueva dirección siguiente
- Apuntador de la entidad a modificar

Para la modificación de la entidad:

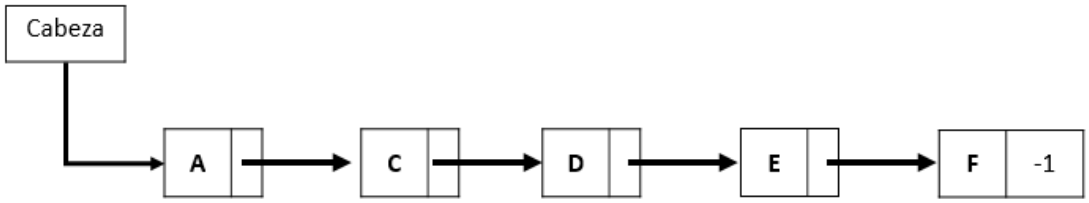
1) Obtener la dirección de la entidad
2) Asignar nuevo nombre
3) Obtener su apuntador anterior
4) Obtener su apuntador siguiente
5) Buscar en su nueva posición (ordenado)
6) Asignar a la nueva dirección la dirección de la entidad
7) La dirección anterior y la anterior siguiente se enlazan
8) Enlazar la nueva entidad con la siguiente dirección de la búsqueda

Casos Importantes al modificar:

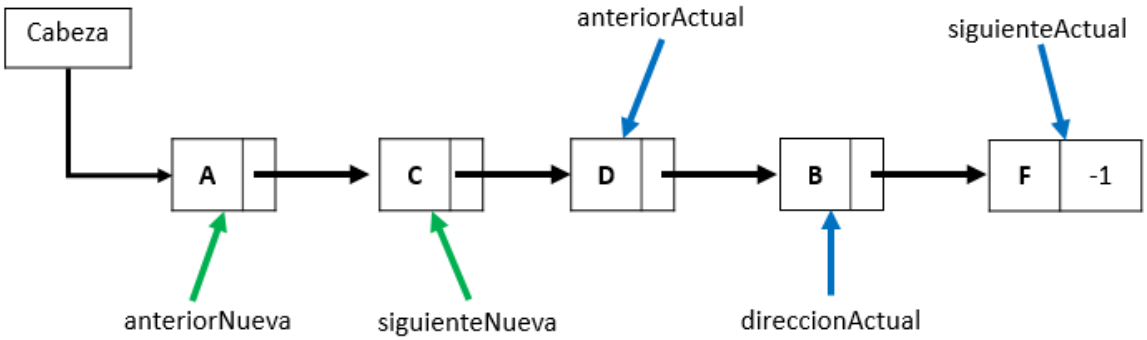
- Si al modificar es la primera entidad
  - Si se modifica a la última entidad
  - Si se modifica a cualquier posición
- Si al modificar es la última entidad
  - Si se modifica a la primera entidad
  - Si se modifica a cualquier posición
- Si al modificar queda en la misma posición

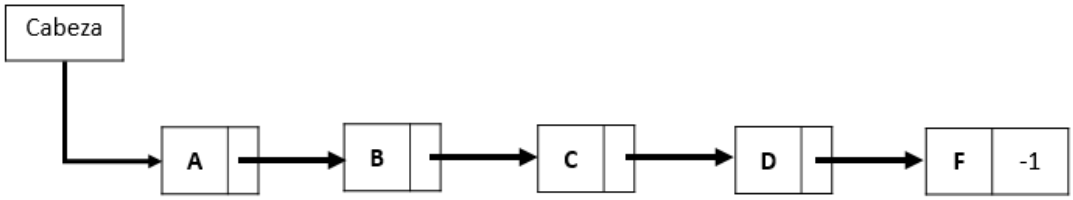
```
            DE = dameDireccionActual(actual);
            br = new BinaryReader(File.Open(nArchivo, FileMode.Open));
            cab = br.ReadInt64();
            br.BaseStream.Position = DE;
            br.ReadString();
            br.ReadUInt64();
            br.ReadUInt64();
            br.ReadUInt64();
            DES = br.ReadInt64();
            br.Close();

            actualSig = actualSiguiente(actual);
            anterior = dameAnterior(nueva,actual); // te da la direccion que te apuntara
            siguiente = dameSiguiente(nueva, actual);  // te da la direcciona a la que apuntaras
            entidadant = dameEntidadAnterior2(actual, DE);
            bw.Close();
            br.Close();
            bw = new BinaryWriter(File.Open(nArchivo,FileMode.Open));

            if (cab == DE)
            {
                bw.BaseStream.Position = 0;
                bw.Write(DES);
                cabecera.Text = DES.ToString();
            }
            if(anterior == 0)
            {
                bw.BaseStream.Position = 0;
                bw.Write(DE);
                cabecera.Text = DE.ToString();

            }
            //actualizar nodo que apunta a nuevo
            bw.BaseStream.Position = anterior + 30 + 8 + 8 + 8;
            if(anterior != 0)
            {
                bw.Write(DE);
            }

            // actualizar nombre
            bw.BaseStream.Position = DE;
            bw.Write(nueva);

            //actualizar  siguiente de nuevo
            bw.BaseStream.Position = DE + 30 + 8 + 8 + 8;
            bw.Write((long)siguiente);
            if(DES !=  siguiente && DES != -1 )
            {
                bw.BaseStream.Position = entidadant + 30 + 8 + 8 + 8;
                bw.Write(DES);
            }
            if(anterior == 0)
            {

            }


            //actualiza nodo
            //bw.BaseStream.Position = entidadant + 30 + 8 + 8 + 8;
            //bw.Write(anterior);
            bw.Close();

        }
        imprimeLista(entidad);
```

Ejemplo gráfico de Modificación:



Modificar (E → B)





Atributos

Es un hecho unitario que caracteriza o describe de alguna manera a una entidad.

Se dice que los atributos son un hecho aislado porque deben ser indivisibles, lo que significa que no pueden dividirse en unidades más pequeñas que tengan algún significado.

- Metadatos del Atributo
- Inserción Atributo
- Eliminación Atributo
- Modificación Atributo

Para poder controlar los Atributos, se utiliza una **Clase Atributo**

```
9   class Atributo
10  {
11      //Declaracion de variavbles
12      private char[] nombre;  // nombre del atributo
13      private char tipo;  // tipo del atributo
14      private int longitud;   // longitud del atributo
15      private long DA;        // Dirección del atributo
16      private int TI;         // Tipo de Indice
17      private long DI;        // Dirección del indice
18      private long DSIG;      // Dirección siguiente atributo
19
20      //Constructor
21      public Atributo(){
22          nombre = new char[30];
23          tipo = ' ';
24          longitud = 0;
25          DA = -1;
26          TI = 0;
27          DI = -1;
28          DSIG = -1;
29      }
30
31      // Get's y Set's
32      public void nombrate(string _nombre)
33      {
34          this.nombre = _nombre.ToCharArray();
35      }
36      public void ponteTipo(char t){
37          this.tipo = t;
38      }
39      public void ponteLongitud(int l){
40          this.longitud = l;
```

METADATOS DEL ATRIBUTO

Nombre
Identificador propio del atributo

Tipo de dato
Se utilizará Entero (E) y Carácter (C)

Longitud de tipo de dato
Representa la longitud en bytes del tipo de dato

Dirección del atributo
Representa la dirección de archivo en donde se almacena el atributo

Tipo de Índice
Identifica el tipo de estructura de datos que se utiliza para representar el índice

- 0 → sin tipo de índice
- 1 → clave de búsqueda
- 2 → índice primario
- 3 → índice secundario
- 4 → Árbol B+

Dirección de índice
Representa la dirección de archivo en donde se comienza a almacenar la estructura de datos utilizada para representar el índice

Dirección del siguiente atributo
Representa la dirección del siguiente atributo que define a la misma entidad.

Nombre	Tipo	Longitud	DA	TI	DI	DSIG

NOMBRE	SIMBOLOGÍA	METODO
Nombre	Nombre	br.ReadString()
Tipo	Tipo	br.ReadChar()
Longitud	Longitud	br.ReadInt32()
Dirección Atributo	DA	br.ReadInt64()
Tipo de Índice	TI	br.ReadInt32()
Dirección Índice	DI	br.ReadInt64()
Dirección Siguiente	DSIG	br.ReadInt64()

Algoritmo: Inserción Atributo

Para poder crear los Atributos e irlos insertando en el diccionario. Se debe conocer la dirección de la Entidad a la que pertenecerá.

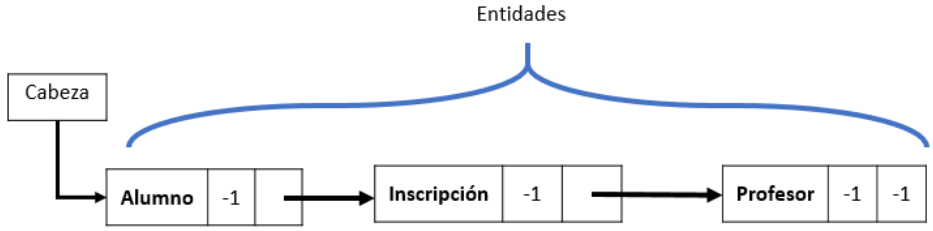
Consideraciones:

- Los Atributos no se insertan en orden (inserción al final).
- El primer Atributo para insertar, su dirección aparecerá en la Entidad.

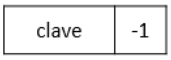
Pasos para insertar Atributo:

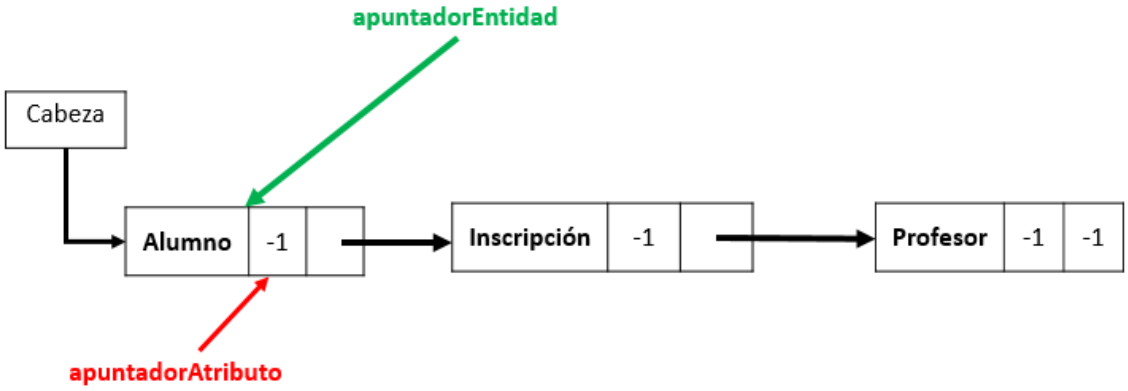
1) Leer la Cabeza del Archivo
2) Posicionar el apuntador en la Entidad que queramos agregar el Atributo
3) Si en el campo de Atributo de la Entidad tiene un -1, entonces no tiene atributos.
4) Buscar su nueva posición e insertar al final del Archivo

Ejemplo Grafico de Inserción de Atributos

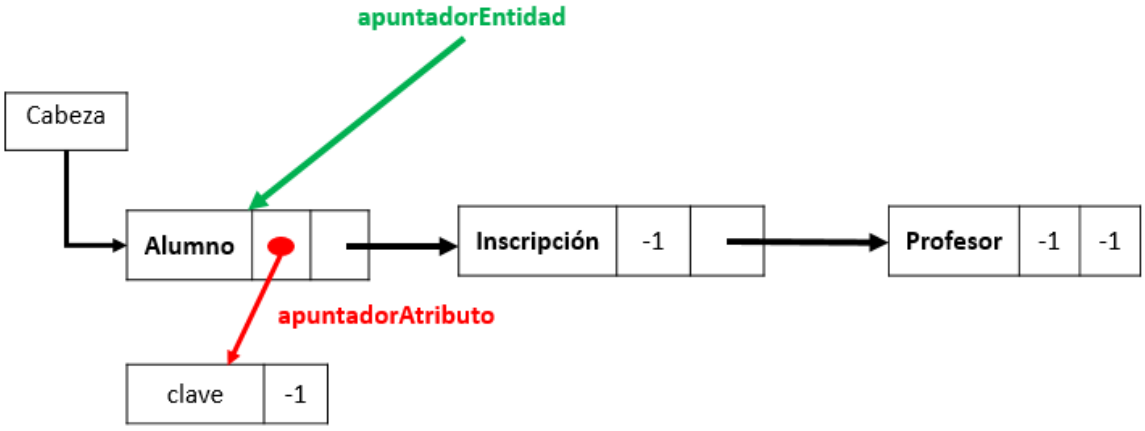


Insertar Atributo (**clave**) en Entidad (**Alumno**)





- Recorre la lista de Entidades hasta encontrar a la que se le va a agregar el Atributo
- Obtener la dirección a la que apunta su campo de Atributo
- Si es -1, se inserta y se modifica la Entidad
- Si no, se inserta al final



```
private void creaAtributo(object sender, EventArgs e)
{

   //bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
   atri = new Atributo();              // Objeto de la clase atributo
   int cont;                           // Variable contador
   _nombre = textBox2.Text;            // Se asigna a la variable nombre, el nombre del atributo dada por el textBox2
   /*****************************************
    * ciclo for que sucede mientras el nombre
    * del atributo tenga una longitud menor a 30
    *****************************************/
    cont = _nombre.Length;
    for (; cont <= 29; cont++)
    {
       _nombre += " ";
    }
   // comboBox5.Items.Add(_nombre);
   /*************************************************************************************
    * Si no hay ningun atributo, entonces se agrega con valores principales en -1
    * SI NO entonces, se añade al final y se acomodan los apuntadores
    *************************************************************************************/
    if (atributo.Count == 0)
    {
       atri.nombrate(_nombre);
       atri.ponteTipo(Convert.ToChar(comboTipo.SelectedItem));
       atri.ponteLongitud(Convert.ToInt32(textBox3.Text));
       atri.direccionate((long)bw.BaseStream.Length);
       atri.ponteTipoIndice((int)tipoIndice);
       atri.ponteDirIndice((long)-1);
       atri.ponteDirSig((long)-1);
       bw.Seek((int)bw.BaseStream.Length, SeekOrigin.Begin);
       bw.Write(atri.dameNombre());
       bw.Write(atri.dameTipo());
       bw.Write(atri.dameLongitud());
       bw.Write(atri.dameDireccion());
       bw.Write(atri.dameTI());
       bw.Write(atri.dameDirIndice());
       bw.Write(atri.dameDirSig());
       //nuevoAtributo = false;

    }
    else
    {
       atri.nombrate(_nombre);
       atri.ponteTipo(Convert.ToChar(comboTipo.SelectedItem));
       atri.ponteLongitud(Convert.ToInt32(textBox3.Text));
       atri.direccionate((long)bw.BaseStream.Length);
       atri.ponteTipoIndice((int)tipoIndice);
       atri.ponteDirIndice((long)-1);
       atri.ponteDirSig((long)-1);

       bw.BaseStream.Position = bw.BaseStream.Length;
       bw.Write(atri.dameNombre());
       bw.Write(atri.dameTipo());
       bw.Write(atri.dameLongitud());
       bw.Write(atri.dameDireccion());
       bw.Write(atri.dameTI());
       bw.Write(atri.dameDirIndice());
       bw.Write(atri.dameDirSig());

    }
    atributo.Add(atri);     //Se añanade el atributo a la lista de atributos

    actualizaEntidad(atri.dameDireccion());
    imprimeLista(entidad);
    imprimeAtributo(atributo);// Se accede al metodo que muestra en el datagrid la lista de atributos
```

Algoritmo: Eliminación Atributo

La eliminación de Atributos requiere el mismo proceso que al eliminar las Entidades solo que primero antes de eliminar el atributo, se debe de buscar su Entidad para poder hacer la reestructuración de su lista de Atributos.

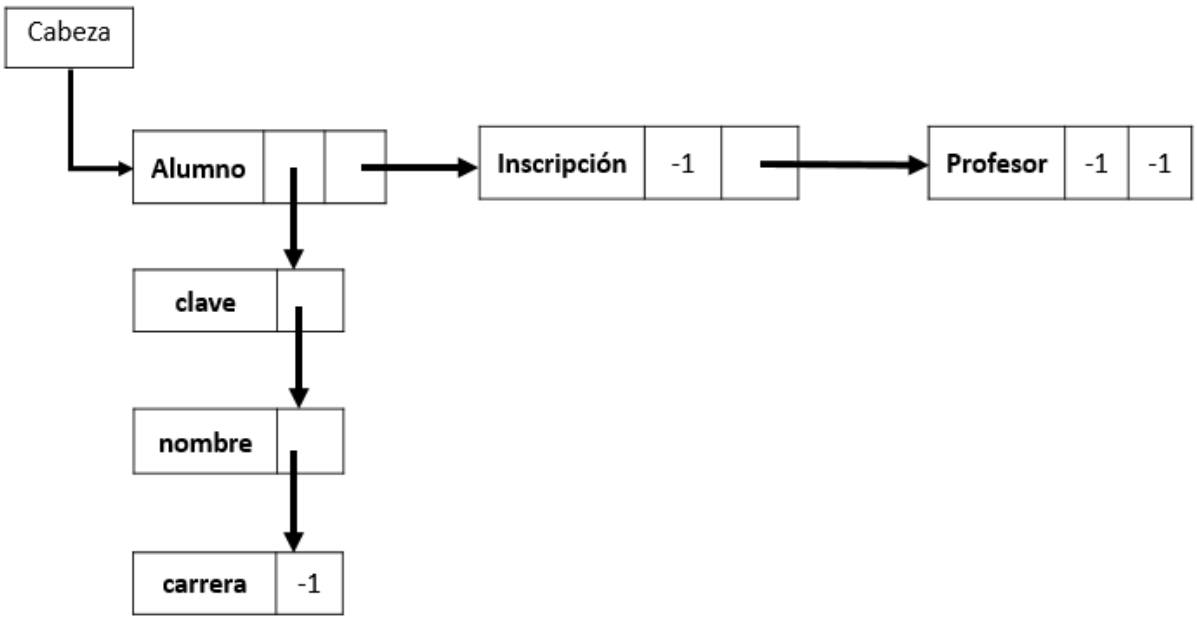
Consideraciones para la eliminación:

- Apuntador a la Entidad
- Apuntador para recorrer la lista de atributos

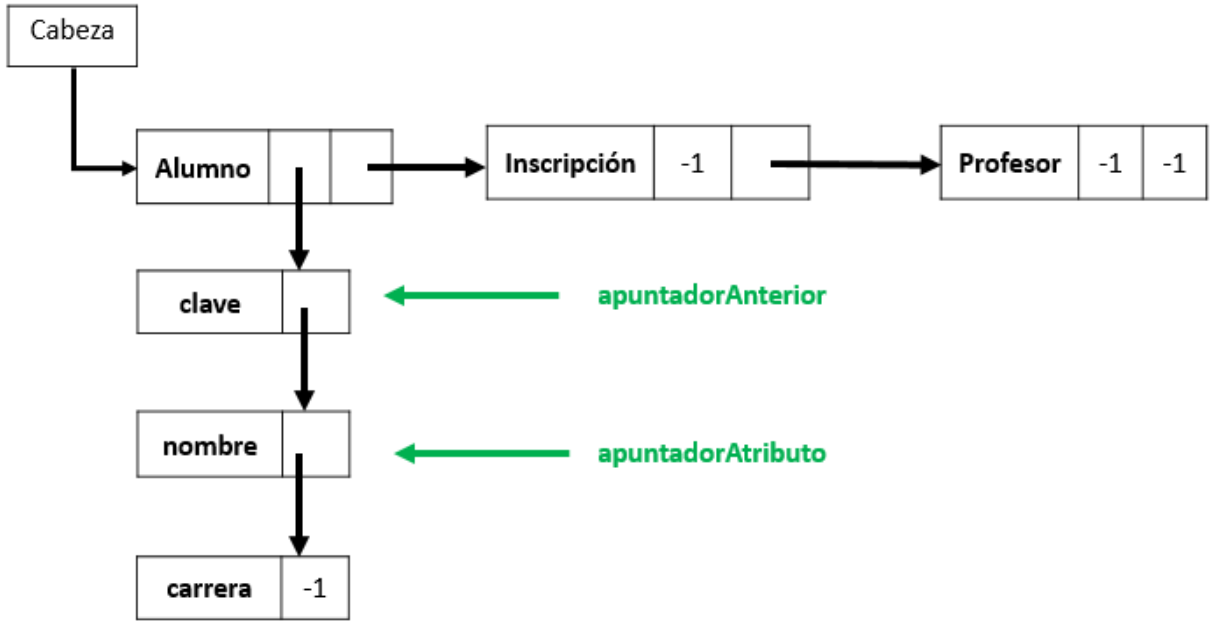
Pasos para la eliminación de Atributos:

1) Leer la Cabeza del Archivo y buscar la Entidad del Atributo a eliminar
2) Recorrer la lista de Atributos hasta encontrar el que se va a eliminar
3) El apuntador anterior del Atributo ahora apunta al siguiente del actual

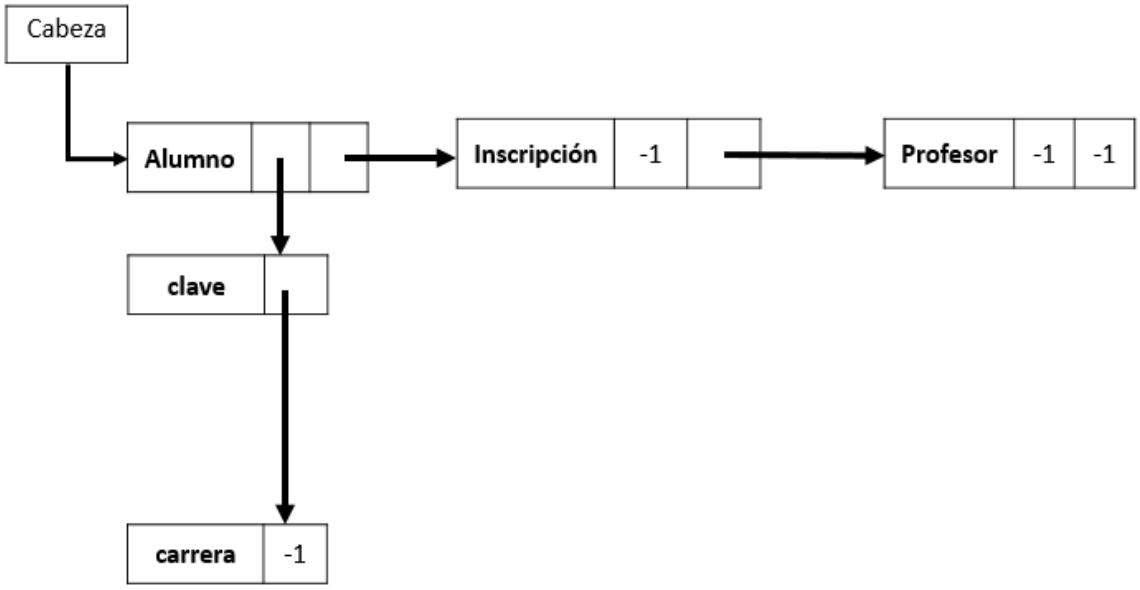
Ejemplo Grafico de Eliminación:



Eliminar Atributo (nombre)




```
apuntadorAnterior.sig = apuntadorAtributo.sig
```



```
/*****************************************************************************************
 *                              METODO   ELIMINA    ATRIBUTO                                 *
 *****************************************************************************************/

private void button4_Click(object sender, EventArgs e)
{
   bw.Close();
   long dAtributo, dEntidad, dir, dSig, anterior;
   string nEntidad, nAtributo;
   string n;
   List<string> entidades = new List<string>();

   foreach(Entidad i in entidad)
   {
       entidades.Add(i.dameNombre());
   }

   EliminaAtributo eliminaAtributo = new EliminaAtributo(entidades, nArchivo);
   if(eliminaAtributo.ShowDialog() ==  DialogResult.OK)
   {
       dAtributo   = eliminaAtributo.dAtributo;
       nAtributo   = eliminaAtributo.nAtributo;
       nEntidad    = eliminaAtributo.nEntidad;

       br = new BinaryReader(File.Open(nArchivo, FileMode.Open));
       br.BaseStream.Position = br.ReadInt64();
       dEntidad = buscaDirEntidad(nEntidad);
       br.BaseStream.Position = dEntidad;
       br.ReadString(); // nombre
       br.ReadInt64();  // dir
       dir = br.ReadInt64();
       br.ReadInt64();
       dSig = br.ReadInt64();
       if(dir == dAtributo) // Es el primer atributo
       {
           //Archivo de lectura abierto
           br.BaseStream.Position = dir;
           br.ReadString(); // nombre
           br.ReadChar();   // tipo
           br.ReadInt32();  // longitud
           br.ReadInt64();  // dirección
           br.ReadInt32();  // tipo indice
           br.ReadInt64();  // dirección indice
           dSig = br.ReadInt64();  // dirección sig
           if(dSig == -1) // Solo hay un atributo
           {
               eliminaPrimero(dEntidad);
               bw.BaseStream.Position = dAtributo;
               string temp = "NULL";
               while (temp.Length <= 29)
               temp += " ";

               bw.Write(temp);

           }
           else // Hay mas de un atributo
           {
               br.Close();
               bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
               bw.BaseStream.Position = dAtributo;
               string temp = "NULL";
               while (temp.Length <= 29)
               temp += " ";

               bw.Write(temp);

               bw.BaseStream.Position = dEntidad + 38;
               bw.Write(dSig);
           }

       }
       else // Es cualquier otro
       {
           br.Close();
           bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
           bw.BaseStream.Position = dAtributo;
```

```
        else // Es cualquier otro
        {
            br.Close();
            bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
            bw.BaseStream.Position = dAtributo;
            string temp = "NULL";
            while (temp.Length <= 29)
            temp += " ";

            bw.Write(temp);

            dSig = dAtributo + 55; //54 ;
                                   //bw.BaseStream.Seek(dAtributo -8, SeekOrigin.Begin);
                                   //  bw.BaseStream.Position = dAtributo - 8;

            anterior = obtenAtributoAnterior(dAtributo, dEntidad);
            bw.BaseStream.Position = anterior + 55;
            dSig = obtenAtributoSiguiente(dSig);
            bw.BaseStream.Position = anterior + 55;
            bw.Write(dSig);
            imprimeLista(entidad);
            imprimeAtributo(atributo);
        }


        // bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
        // bw.BaseStream.Position = dAtributo;


    }
    imprimeLista(entidad);
    imprimeAtributo(atributo);


}
```

Algoritmo: Modificar Atributo

A diferencia del algoritmo de modificación de Entidades, el de Atributo es más sencillo. Ya que no se requiere ordenar. Simplemente se encuentra el atributo y se modifican los campos.

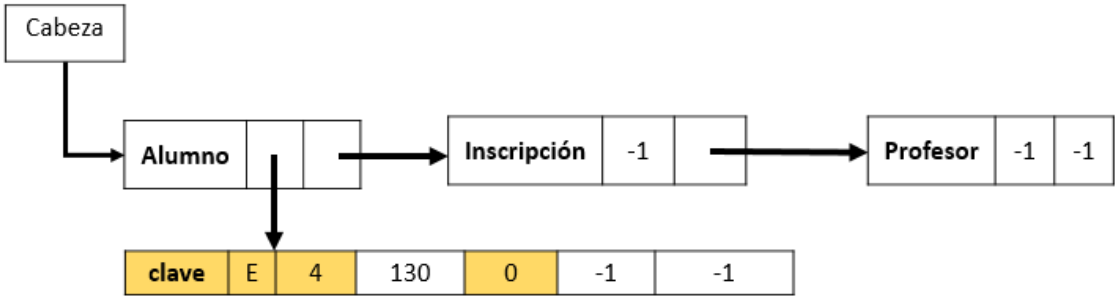
Importante

Una vez que ya existan registros no se debe de modificar los atributos ya que eso altera todos los índices y registros.

Pasos para la modificación de Atributos

1) Buscar la entidad del Atributo
2) Recorrer la lista de Atributos hasta encontrar el deseado
3) Modificar sus valores

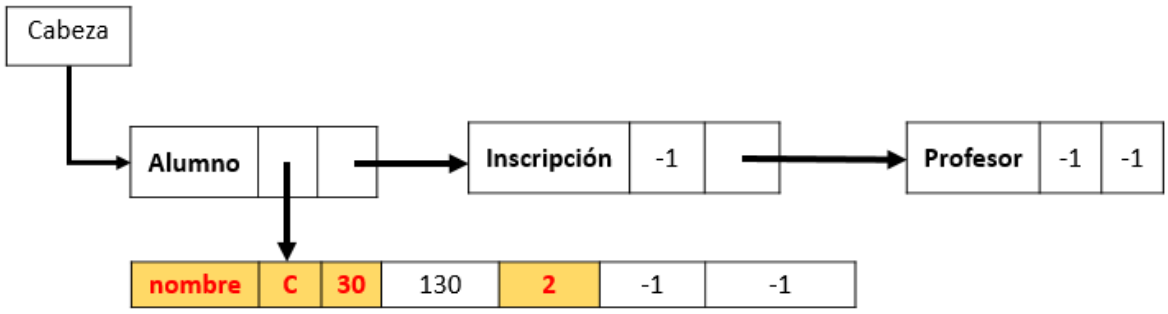
Ejemplo Gráfico de Modificación



Modificar Atributo (clave) por lo siguiente:

- Nombre = nombre
- Tipo = C
- Longitud = 30
- Índice = Índice Primario (2)

Las direcciones no se pueden modificar.



```
private void modificaAtributo(object sender, EventArgs e)
{
    ModificaAtributo mod = new ModificaAtributo(atributo);
    Atributo nuevo;
    bw.Close();
    br.Close();
    if(mod.ShowDialog() == DialogResult.OK)
    {
        nuevo = mod.dameNuevo();
        bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
        bw.BaseStream.Position = nuevo.dameDireccion();
        bw.Write(nuevo.dameNombre());
        bw.Write(nuevo.dameTipo());
        bw.Write(nuevo.dameLongitud());
        bw.Write(nuevo.dameDireccion());
        bw.Write(nuevo.dameTI());
        bw.Write(nuevo.dameDirIndice());
        bw.Write(nuevo.dameDirSig());
        bw.Close();
        imprimeLista(entidad);
        imprimeAtributo(atributo);
    }
}
```

Ejemplo Diccionario de Datos

Entidad: Alumno

Atributos:

- Cve_unica E 4 2
- Nombre C 15 1
- Cve_carrera E 4 3
- Genero C 1 0

8 | Alumno	**8**	**70**	**-1**	**-1**		
cve_única	E	4	70	2	-1	133
nombre	C	15	133	1	-1	196
cve_carrera	E	4	196	3	-1	259
genero	C	1	259	0	-1	-1

Registros

Un registro es un conjunto de campos que contienen los datos que pertenecen a una misma repetición de entidad. Se le asigna automáticamente un número consecutivo (número de registro) que en ocasiones es usado como índice, aunque lo normal y práctico es asignarle a cada registro un campo clave para su búsqueda.

- Inserción
- Eliminación
- Modificación

Tabla de Registros

Cada Entidad que se crea genera dos archivos, uno para los registros y otro para los índices, así, si la Entidad se llama Alumno, entonces habrá dos archivos de nombre **Alumno.dat** y **Alumno.idx.**

Los metadatos de un registro dependen de los Atributos que tenga su Entidad, así que la tabla de registros es **dinámica** ya que no siempre serán los mismos.

Dirección	Atributos	Dirección Siguiente

- Para obtener el tamaño de un registro, se debe sumar la longitud de todos los atributos que tiene la entidad y su dirección con su apuntador al siguiente Registro
- La Entidad en su metadato de registros, apunta al primer registro en el archivo .**dat**
- En la tabla de registros, las columnas [0] y [n-1] son la dirección y el apuntador al siguiente registro, respectivamente.

Es muy importante tener bien estructurado los registros, ya que para todas las organizaciones de los índices se ocupará esta tabla de registros.

Algoritmo: Inserción de Registros

La inserción de registros, al igual que las Entidades y los Atributos es un algoritmo sencillo de búsqueda hasta encontrar el ultimo registro.

Pero, si un atributo es **clave de búsqueda**, entonces los registros se insertarán en orden de acuerdo con esa clave.

La dirección de cada registro siempre el tamaño del archivo hasta ese momento.

Pasos para la inserción de Registros

1) Buscar la Entidad en la lista de archivos para manipular el .**dat**
2) Preguntar si la Entidad tiene registros (el metadato es diferente de -1)
3) Si tiene registros:
    a. Se recorren los registros hasta encontrar el ultimo, este apunta a -1.
    b. Enlazar el anterior registro al nuevo
4) Si no tiene registros:
    a. Insertar en el archivo
    b. Agregar su dirección al metadato de su Entidad

Tipos de inserción ordenada

- Si la clave de búsqueda es una cadena
    - Se utiliza el método **compareTo** para posicionarlas (página 10)
- Si la clave de búsqueda es numérica
    - Se utiliza una comparación de mayor, menor, igual (>, < , ==)

Algoritmo: Eliminación Registros

La eliminación de Registros requiere de la misma búsqueda implementada para atributos y entidades.

Consideraciones al Eliminar:

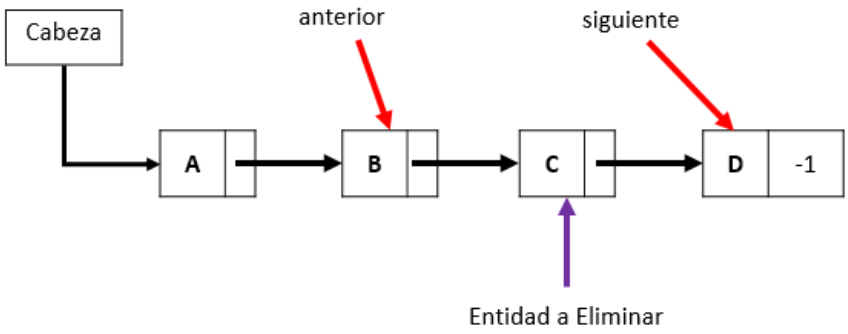
- Se va a eliminar el primer registro
- Se va a eliminar cualquier registro
- Se elimina el único registro

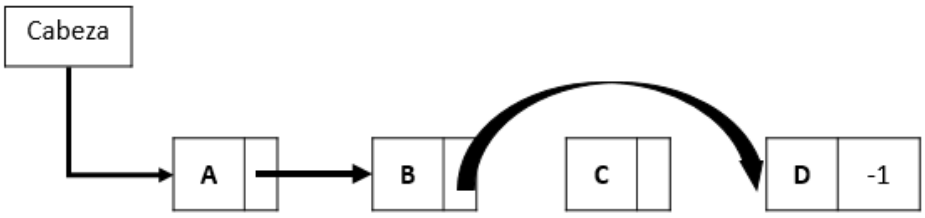
Pasos para la eliminación de Registros

1) Obtener de la Entidad el primer apuntador del registro
2) Recorrer la lista de registros hasta encontrar el registro a eliminar
3) Hacer que el registro anterior ahora apunte al siguiente del actual
4) Si se elimina el primer registro se hace lo mismo del paso 3 pero ahora es desde el apuntador de la Entidad

Al eliminar los registros, no se borran del archivo, solamente se deja de apuntar a ellos.

Eliminar (C)





Algoritmo: Modificación de Registros

De todos los algoritmos de Registros, el de modificación es el más complejo ya que requiere de 4 apuntadores además que tiene un mínimo de 4 casos para la modificación.

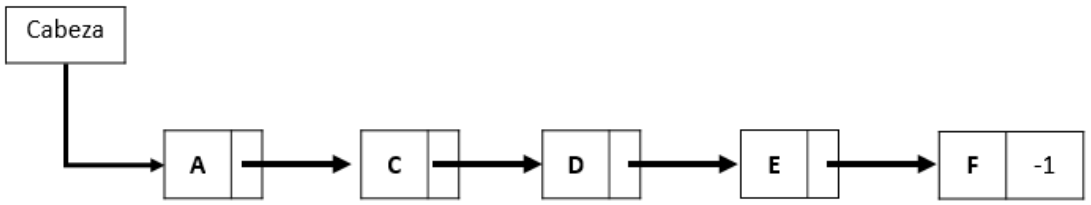
Consideraciones al eliminar

1) El registro que se va a modificar es el primero
2) El registro que se va a modificar es el ultimo
3) El registro que se va a modificar esta en cualquier posición
4) El registro modificado quedara en la primera posición
5) El registro modificado quedara en la última posición
6) El registro modificado quedara en cualquier posición

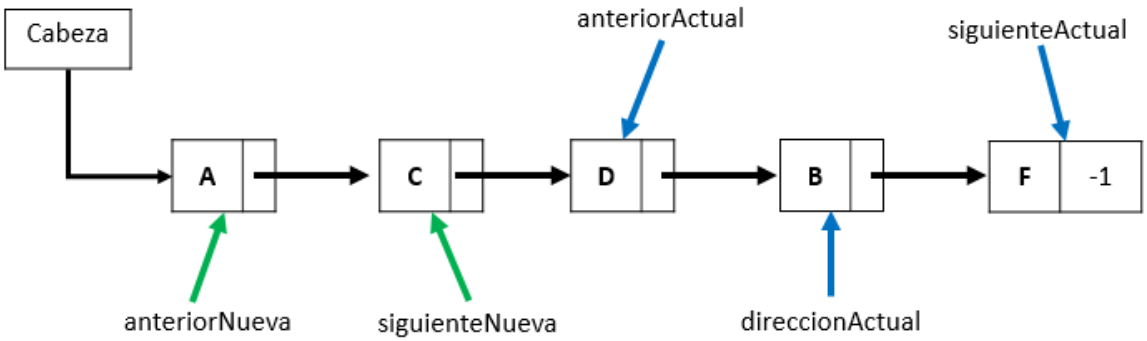
Pasos para la modificación de registros

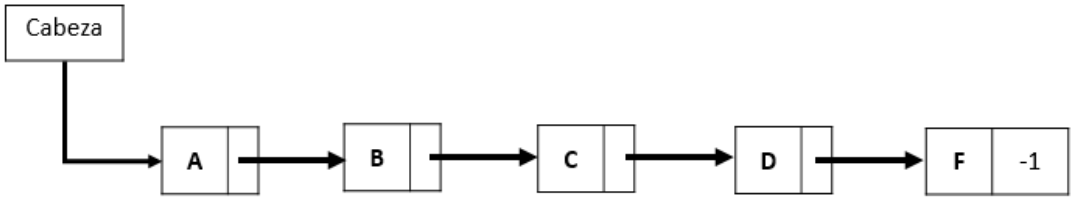
1) Posicionar el primer apuntador del registro dado por la Entidad
2) Recorrer los registros hasta encontrar el deseado a eliminar
3) Modificar los campos del registro
4) Obtener dirección actual
5) Obtener su dirección anterior de su posición actual
6) Obtener su dirección siguiente de su posición actual
7) Recorrer la lista hasta encontrar su nueva posición
8) Obtener su dirección anterior de la nueva posición
9) Obtener su dirección siguiente de la nueva posición
10) Enlazar el anterior actual con el siguiente actual
11) Enlazar el anterior nuevo con la dirección del registro modificado
12) Enlazar la dirección del registro modificado con la dirección siguiente nueva

Ejemplo gráfico de Modificación:



Modificar (E → B)





Ejemplo Tabla de Registros

- Clave de búsqueda es **nombre**
- La Entidad Alumno, en su metadato apunta a la dirección del primer Registro

Alumno	8	70	80	-1

Alumno.dat

DR	cve_única	nombre	cve_carrera	Genero	DSIG
0	830022	JOSE	15	M	-1
40	292122	JORGE	14	M	0
80	993323	ABRAHAM	15	M	40

Índices

Se utiliza este tipo de organización de archivo cuando existe la necesidad tanto de acceder a los registros secuencialmente por algún valor de la clave de búsqueda (llave primaria o secundaria), como de accesarlos individualmente.

- Índice Primario
- Índice Secundario
- Árbol B+

Organización Indexada

Para una mejor forma de acceder a todos los registros, se utilizan los índices, el proyecto cuenta con 3 tipos de índices y la forma en la que se pueden ir creando y agregando a los índices es mediante la tabla de registros.

Dirección	Atributos	Dirección Siguiente

Recordando que un registro es dinámico pero una vez que se crea su tabla, seguirá siendo la misma.

Cada columna de la tabla especifica un atributo y a su vez cada atributo puede tener un índice entonces:

	Dirección Registro	clave	nombre	carrera	Dirección Siguiente

Considerando lo siguiente:

Atributo (clave con Índice Primario (2))

Atributo (nombre con clave de Búsqueda (1))

Atributo (carrera con Índice Secundario (3))

La posición de la columna nos puede decir cuál es la clave que se insertará según el índice.

[0] [1] [2] [3] [4]

Donde:

[0] → Dirección Registro

[1] → clave

[2] → nombre

[3] → carrera

[4] → Dirección Siguiente

Las columnas [1] y [3] son índice primario y secundario respectivamente.

Entonces, para poder insertar las claves a los índices dependiendo de cuál sea su índice, se tiene que leer la tabla de registros y así cuando se agregue un registro, se obtiene su columna [1] y se agrega a su índice primario.

Reservación de Espacio para Índices

Todos los índices de una entidad se guardarán en un archivo .**idx**, así si la entidad se llama Alumno, su archivo de índice será **Alumno.idx**

A diferencia del diccionario de datos y los registros, los índices ya deben de contar con un espacio de almacenamiento definido.

Índice Primario

Un índice primario, lleva 2 campos: la clave de búsqueda y su dirección en la tabla de registros.

Para que sea índice primario deben ser claves únicas.

clave	dirección

Donde:

[clave] → Su longitud es la que se estableció en el Atributo

[dirección] → Su longitud es de 8 bytes (`long`)

Considerando que clave tiene una longitud de 4 bytes, entonces el renglón del índice primario tiene una longitud de 12 bytes. Ahora para generar el espacio se deben considerar un numero n de claves. Suponiendo que se tomaran 10 claves.

El bloque de memoria para el índice primario será de 120 bytes

clave	dirección

Índice Secundario

El índice secundario si permite claves repetidas por lo que su estructura es una matriz donde la primera columna es la clave y todas las demás columnas son las direcciones de los registros que tienen esa clave.

Para poder generar el espacio se debe obtener la longitud del atributo que es índice secundario. Suponiendo que se usar el Atributo (nombre con longitud 30) como índice secundario.

Se reserva un espacio de m x n de 10 claves.

nombre	dir	dir	dir	dir	dir	dir	dir	dir	dir	dir

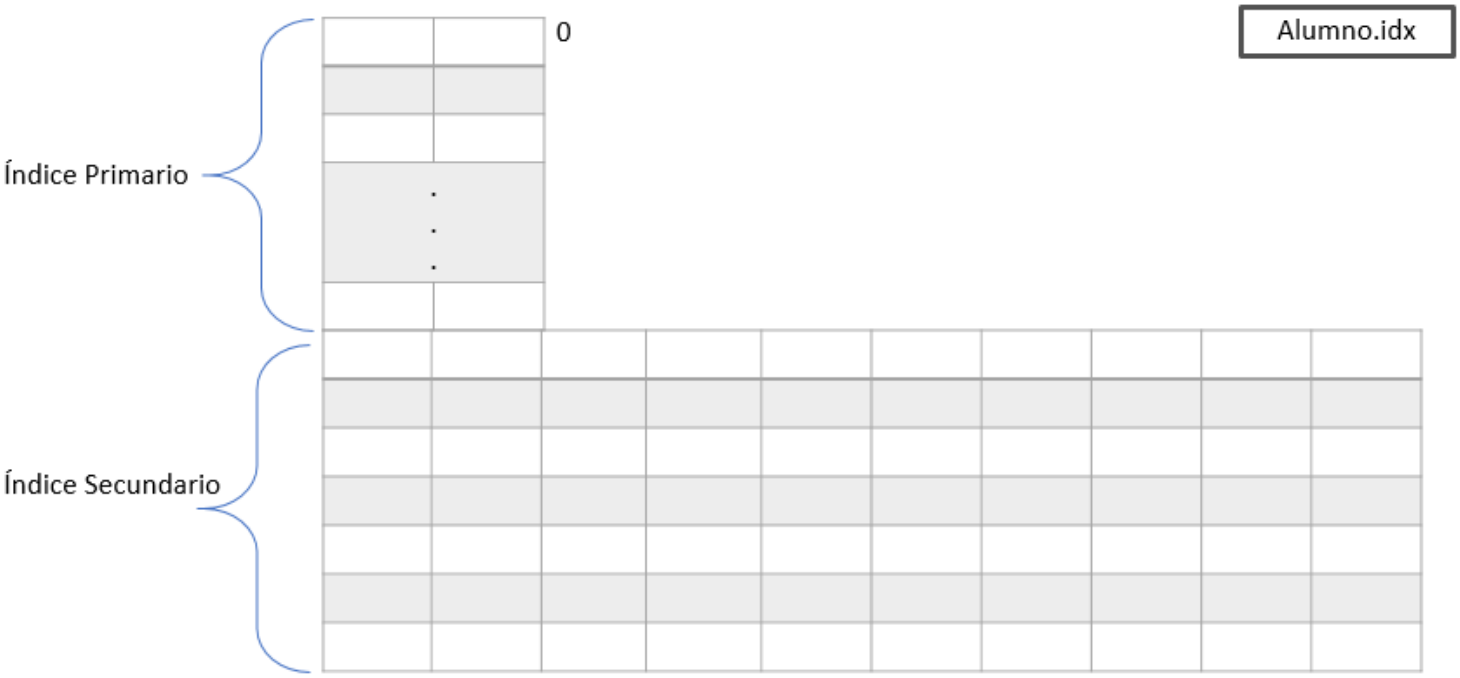
[nombre] → 30 bytes

[dir] → 8 bytes

El renglón tiene una longitud de (8 bytes * 10) + 30 = 110 bytes

entonces el bloque completo tendrá un tamaño de (110 bytes * 10) = 1,100 bytes

Ejemplo gráfico de Archivo .idx



Inserción Índice Primario

Pasos para la inserción al índice primario

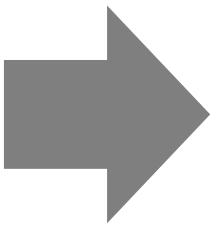
1) Al insertar el registro obtener su columna [n] que es su clave de índice primario
2) Buscar en su atributo la dirección índice donde está el bloque del primario
3) Recorrer las claves hasta encontrar su posición
4) Insertar junto a su dirección del registro

```
bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
Primario p;
List<Primario> primario = new List<Primario>();

for(int i = 0; i < dataGridView4.Rows.Count-2; i++)
{
 p = new Primario();
 p.dirVal = Convert.ToInt64(dataGridView4.Rows[i].Cells[0].Value);
 p.val = Convert.ToInt32(dataGridView4.Rows[i].Cells[posPrimario+1].Value);
 primario.insertaOrdenado(p);

}
```

Ejemplo grafico

Insertar clave (44)

32	2351
55	432
77	1001



32	2351
44	534
55	432
77	1001

Inserción Índice Secundario

Pasos para la inserción del índice secundario

1) Obtener la celda [i, j] de la tabla de registros
2) Buscar la dirección del bloque secundario en su atributo
3) Buscar si existe la clave
4) Si existe se agrega solo la dirección en el renglón donde está la clave
5) Si no existe se agrega un nuevo renglón con la clave y su dirección

```
Atributo iSecundario = new Atributo();
Atributo iPrimario = dameIndicePrimario(comboBox6.Text);
int pos = 0;
poSI = 0;
int j = 0;
for (int i = 0; i < dataGridView3.Columns.Count; i++)
{
  if(j < secundarios.Count)
  if(dataGridView3.Columns[i].Name == secundarios[j])
  {
    iSecundarios.Add(i);
    if (secundarios[j] == comboSecundario.Text)
    {
      poSI = TAMSEC *j;
      iSecundario = dameIndiceSecundario(comboBox6.Text,secundarios[j]);
      bw = new BinaryWriter(File.Open(nArchivo, FileMode.Open));
      if (iPrimario != null)
            TAMPRIM = dameTamPrimario();
      else
            TAMPRIM = 0;
      bw.BaseStream.Position = iSecundario.dameDireccion() + 30 + 1 + 4 + 8 + 5;
      if (j == 0)
        bw.Write(Convert.ToInt64(TAMPRIM));
      else
        bw.Write(Convert.ToUInt64(TAMSEC * j));
      bw.Close();
    }
    j++;
  }
}

```

Visualización de Registros e índices

REGISTRO DE DATOS

8 | Alumno	8	70	80	-1

Alumno.dat

DR	cve_única	nombre	cve_carrera	Genero	DSIG
0	830022	JOSE	15	M	-1
40	292122	JORGE	14	M	0
80	993323	ABRAHAM	15	M	40

Alumno.idx

292122	40
830022	0
993323	80

14	40			
15	0	80		

Árbol B +

La estructura de índice de árbol B+ es la más entendida de las estructuras de índices que mantienen su eficiencia a pesar de la inserción y borrado de datos. Un índice de árbol B+ toma la forma de un árbol equilibrado donde los caminos de la raíz a cada hoja del árbol son de la misma longitud.

Estructura de un árbol B+

P1	K1	P2	K2	. . .	Pn-1	Kn-1	Pn

Nodo de un árbol B+

- En la figura se muestra un nodo típico de un árbol. Puede contener hasta n-1 claves de búsqueda (k1, k2, . . ., kn-1) y n apuntadores (P1, P2, . . ., Pn). Los valores de las claves de búsqueda se mantienen ordenados; así, si **i < j**, entonces **ki < kj**.
- Considere primero la estructura de los nodos hoja.
- Para i = 1, 2, …,n-1, el apuntador Pi apunta, o bien a un registro del archivo con valor de clave de búsqueda ki, o bien a una lista de apuntadores, cada uno de los cuales apunta a un registro del archivo con valor de la clave de búsqueda ki.

Nodo completo de un árbol B+

DN	TN	AP1	CB1	AP2	CB2	AP3	CB3	AP4	CB4	AP5

Donde:

- DN → Dirección del nodo
- TN → Tipo de Nodo
- AP → Apuntador
- CB → Clave de búsqueda

Tipos de Nodo

- H → Hoja
- I → Intermedio
- R → Raíz

Tamaño de Nodo

DN	TN	AP1	CB1	AP2	CB2	AP3	CB3	AP4	CB4	AP5
8	1	8	4	8	4	8	4	8	4	8

TAM NODO = 65 bytes

Inserción Árbol B+

Consideraciones para inserción

- Si no tiene nodo raíz
- Si tiene raíz y nodo intermedio
- Si tiene raíz y no tiene nodo intermedio
- Si al querer insertar se llena un nodo hoja
- Si al agregar otro nodo hoja se llena la raíz

Inserción en nodo Hoja

1) Recorrer el nodo hasta encontrar la posición
2) Si el nodo tiene espacio para la clave, insertar
3) Si el nodo no tiene espacio (nodo.Count == 4) entonces
    a. Crear un nuevo nodo Hoja y agregar las claves [2][3][4] al nuevo nodo y eliminar las claves [2][3] del nodo anterior
    b. b Agregar la primera clave y la dirección del nodo al nodo padre (Intermedio o Raíz)

Inserción en nodo Raíz

1) Recorrer el nodo hasta encontrar la posición
2) Agregar si es Hoja la clave [0] del nuevo nodo y la dirección del nodo
3) Agregar si es Intermedio la clave [2] y la dirección del nuevo nodo
4) Si no se puede agregar entonces
    a. Crear un nuevo nodo, el nuevo y el actual se convierten en Intermedio
    b. El nodo actual conserva las claves ordenadas [0] [1]
    c. El nodo nuevo se le añade las claves [3] [4]
    d. Se crea nodo raíz y se añade la clave [2] con las direcciones de los 2 nodos intermedios

Inserción en nodo Intermedio

1) Recorrer el nodo hasta encontrar la posición
2) Si el nodo tiene espacio agregar clave y dirección
3) Si no tiene espacio entonces crear otro nodo Intermedio
    a. En el primer nodo se quedan las claves [0] [1]
    b. En el nuevo nodo se añaden las claves [3] [4]
    c. Se añade la clave [2] al nodo Raíz con la dirección del nuevo nodo Intermedio

Algoritmo busca Nodo

- Recibe como parámetro la clave que se añadirá
- Regresa la dirección del nodo en que se debe insertar

```
public long buscaNodo(int clave)
{
    bool intermedio = false;
    intermedio = existeIntermedio();
    int total = totalIntermedio();
    int cont = 0;
    Nodo inter, inter2 = new Nodo();
    foreach (Nodo n in this)
    {
        if (n.tipo == 'R' && intermedio == false)
        {
            for (int i = 0; i < n.clave.Count; i++)
            {
                if (clave < n.clave[i])
                {

                    return n.direccion[i];
                }

                else if (i == n.clave.Count - 1)
                {
                    return n.direccion[i+1]; // i
                }


            }
        }
        else if (n.tipo == 'I' && intermedio == true)
        {
            inter = dameNodo(dameRaiz());
            cont++;
            for (int i = 0; i < inter.clave.Count; i++)
            {
                if (clave < inter.clave[i])
                {
                    //inter2 = inter;
                    inter2 = dameNodo(inter.direccion[i]);
                    break;
                }
                else if (i == inter.clave.Count - 1)
                {
                    inter2 = dameNodo(inter.direccion[i + 1]);
                    break;
                }



            }
            if (inter2.clave.Count > 0)
            {
                for (int i = 0; i < inter2.clave.Count; i++)
                {
                    if (clave < inter2.clave[i])
                    {

                        return inter2.direccion[i];
                    }

                    else if (i == inter2.clave.Count - 1)
                    {
                        return inter2.direccion[i + 1];
                    }


                }
            }

        }
}

/*
```

Algoritmo agrega Dato

- Recibe la dirección, clave del Registro y la dirección del nodo a insertar
- Devuelve un booleano si no se pudo agregar porque está lleno el nodo

```
public bool agregaDato(long direccion, int clave, long apuntador)
{
    int g;
    int pos = 0;
    long raiz = dameRaiz();
    List<int> tempClave = new List<int>();
    List<long> tempDir = new List<long>();
    foreach (Nodo np in this)
    {
        if (np.dirNodo == direccion)
        {
            if (np.clave.Count < 4)
            {

                if (raiz == direccion)
                {
                    np.clave.Add(clave);
                    np.direccion.Add(apuntador);
                    ordenaRaiz();
                }
                else if (np.tipo == 'I')
                {
                    np.clave.Add(clave);
                    np.direccion.Add(apuntador);
                    ordenaIntermedio(direccion);
                }
                else
                {
                    np.clave.Add(clave);
                    np.direccion.Add(apuntador);
                    ordenaNodo(np);

                }
                return false;

            }
            else if (np.clave.Count == 4) // Aqui se reestructura el nodo
            {

                return true;

            }

        }
    }
    return false;
}
```

Eliminación de árbol B+

Consideraciones para inserción

- Si no tiene nodo raíz
- Si tiene raíz y nodo intermedio
- Si tiene raíz y no tiene nodo intermedio
- Si al querer eliminar quedan menos n/2 claves
- Si al querer eliminar necesita pedir claves
- Si al querer eliminar necesita fusionarse con otro nodo

Pasos para eliminar nodo hoja

1) Recorrer el nodo hasta encontrar el nodo
2) Eliminar clave
3) Si el número de claves es mayor o igual a n/2 entonces continuar
4) Si el número de claves es menor a n/2 entonces
    a. Buscar si tiene nodos hermanos (izquierda o derecha)
    b. Preguntar si pueden pasarle una clave siempre y cuando se cumpla la regla de que no se quede con menos de n/2
    c. Si le pueden prestar claves entonces
        I. Si es nodo izquierdo el que le va a prestar agregar al nodo actual la **última** clave al nodo actual y se modifica el Intermedio con la clave recientemente agregada
        II. Si es nodo derecho entonces agregar a la **primera** clave al nodo actual y se modifica el intermedio con la clave recientemente agregada
    d. Si no le pueden prestar claves entonces
        I. Fusionar con el nodo (izquierda o derecha)
        II. Si se fusiona con el nodo izquierdo entonces se agregan las claves del nodo actual al izquierdo y se elimina el apuntador del nodo actual al intermedio
        III. Si se fusiona con el nodo derecho entonces se agregan las claves del nodo derecho al nodo actual y se elimina el apuntador del nodo derecho que tiene el nodo Intermedio

Pasos de eliminación nodo Intermedio

1) Verificar si su hermano (izquierdo/ derecho) le pueden prestar claves
2) Si le presta nodo Izquierdo
    a. Agregar Ultima clave con su dirección al nodo actual y la clave que está en Raíz se sustituye por la clave que se añadió
3) Si le presta nodo Derecho
    a. Agregar primera clave con su dirección al nodo actual y la clave que está en Raíz se sustituye por la clave que se añadió
4) Se tiene que fusionar
    a. Toma el otro intermedió y se fusionan las claves además de añadirse la clave de nodo Raíz, ahora el nodo que tiene la fusión se convierte en nodo Raíz

Algoritmo Fusiona Derecha

- Recibe la dirección del nodo actual, nodo derecho y la dirección de su nodo intermedio o raíz
- Devuelve un booleano si el nodo intermedio o raíz se queda con menos de n/2 claves

```
public bool fusionDerecha(long actual, long derecha, long inter)
{
    bool band = false;
    Nodo nActual = dameNodo(actual);
    Nodo dDerecha = dameNodo(derecha);
    Nodo intermedio = dameNodo(inter);
    long izquierda, der;
    for (int i = 0; i < dDerecha.clave.Count; i++)
    {
        agregaDato(actual, dDerecha.clave[i], dDerecha.direccion[i]);
    }
    // ELIMINA LOS DATOS DEL NODO DERECHO
    dDerecha.clave.RemoveAt(1);
    dDerecha.clave.RemoveAt(0);
    dDerecha.direccion.RemoveAt(1);
    dDerecha.direccion.RemoveAt(0);
    //elimina del nodo intermedio
    bool band2 = eliminaIntermedio(inter, derecha);
    if (band2 == true)
    {
        return true;
    }
    else
        return false;

}
```

Algoritmo elimina Raíz

- Recibe dirección de la raíz o nodo intermedio y el nodo izquierdo del actual
- Regresa variable booleana si la raíz se quedó sin claves o no

```
public bool eliminaRaizIzquierdo(long inter, long izquierdo)
{
    Nodo nRaiz = dameNodo(dameRaiz());
    int c = 0;
    foreach (long cont in nRaiz.direccion)
    {
        if (cont == inter)
        {
            nRaiz.clave.RemoveAt(c - 1);
            nRaiz.direccion.RemoveAt(c);
            break;
        }
        c++;
    }
    if (nRaiz.clave.Count == 0)
    {
        Remove(nRaiz);
        return true;
    }
    else
        return false;
}
```