

Práctica 2b

23 de mayo

2010

Autor: José Ángel García Fernández

Organización y
Gestión de
Archivos

Memoria Práctica 2b

Diseño y tipos de datos

He usado los ya creado en las prácticas anteriores, añadiendo alguna funcionalidad más que en un primer momento parecía que no hacía falta, pero al final si era necesaria, como por ejemplo añadirle la **sobrecarga de operadores > y <** al tipo **Registro** para facilitar la comparación y ordenación de los bloques. Destacar que muchos de los métodos de la clase original **ESFicheroBinario** hacían referencia a registros, ahora en la clase del archivo de bloques también se llamarán así debido a la herencia, pero no trabajaremos con los registros sino con el bloque completo, simplemente usando el tamaño del bloque y no el del registro.

Tipo para representar el bloque

Declaramos un **struct** que contiene los campos del Bloque. Modelamos la posición del siguiente bloque en la lista como un **long**, el numero de registros como un **int** y la cadena que contendrá a los registros como un **char***, sin tamaño predefinido ya que viene impuesta por el parámetro **n** (factor de blocaje).

```

6      typedef struct
7      {
8          long posSig;
9          int nRegs;
10         char* cadRegs;

```

Clase BloqueIND

```

14     class BloqueIND
15     {
16     private:
17         BLOQUE bloque;
18         friend class INDICEND;
19         friend class FileBloques;
20     public:
21         BloqueIND(int tamMaxBloque, int sig=-1);
22         ~BloqueIND();
23         //Añade un registro al bloque
24         int addReg(Registro&);
25         //Resetea el bloque
26         void clear();
27         //Permite obtener el registro i del bloque de zona maestra
28         int getReg(int i, Registro&);
29         //Permite obtener el ultimo registro del bloque de zona maestra
30         int getUltimoReg(Registro&);
31         //Busca el registro con clave primaria valor en el bloque de zona maestra
32         bool buscar(char* valor);
33         //Convierte el bloque en un vector de registros para facilitar insercion
34         void toVector(vector<Registro*>);
35         //Permite eliminar el ultimo registro del bloque de zona maestra
36         int eliminarUltimoReg(int maxRegs);
37         //Obtiene el valor de la cadena para guardar la información en el STRUCT.
38         void ObtenerCadenaBloque(char* cadenaRegistro);
39         //Establece el valor de la cadena en base a la información del STRUCT.
40         int EstablecerCadenaBloque(char* cadenaRegistro);
41         //Muestra el bloque
42         void mostrar();
43         //Muestra el bloque en formato condensado
44         friend ostream& operator<<(ostream&, const BloqueIND&);
45         //Sobrecarga del operador ++ para aumentar el nRegs
46         BloqueIND& operator++(int);
47         //Sobrecarga del operador =
48         BloqueIND& operator=(const BloqueIND&);
49     };

```

Esta clase envolverá a la estructura **BLOQUE** añadiéndole las funcionalidades que necesitaremos. Destacar que declaramos amigas a **INDICEND** y a **FileBloques** para poder acceder a la propiedad **bloque**.

Métodos BloqueIND

Añadir registros al bloque

```

431 int BloqueIND::addReg(Registro& reg)
432 {
433     //si es el primero lo copio sino lo concateno
434     char* buffer= new char[100];
435     reg.EstablecerCadenaRegistro(buffer);
436     if (bloque.nRegs==0)strcpy(bloque.cadRegs, strcat(buffer, "\n"));
437     else strcat(bloque.cadRegs, strcat(buffer, "\n"));
438     bloque.nRegs++; //aumenta nRegs
439     delete buffer;
440     return strlen(bloque.cadRegs);
441 }

```

Este método permite introducir un nuevo registro en el bloque, usando el método **EstablecerCadenaRegistro()** y teniendo en cuenta si es el primer registro o no, devolvemos la longitud actual del bloque.

Buscar registro en bloque

```

485 bool BloqueIND::buscar(char* valor)
486 {
487     Registro reg;
488     bool encontrado=false;
489     vector<string> aux;
490     string copia(bloque.cadRegs);
491     Tokenize(copia,aux,"\n");
492     for (int i=0;i<bloque.nRegs;i++)
493     {
494         //busca en el bloque si esta el registro con clave valor
495         reg.ObtenerCadenaRegistro(const_cast<char*>(aux[i].c_str()));
496         if (strcmp(reg.Bridge.Id,valor)==0)
497         {
498             cout<<reg<<endl;
499             encontrado=true;
500         }
501     }
502     return encontrado;
503 }

```

Con este método buscamos en el bloque el registro con clave valor, para ello hacemos uso de la función **Tokenize** que partirá el bloque en **subcadenas** que representarán cada registro, para que en [492,501] podamos ir buscando el registro que cumpla la condición, si lo encontramos lo mostramos.

Clase INDICEND

```

51 class INDICEND:public ESFicheroBinario
52 {
53 private:
54     tipoIndice tI; //vale 1 o 2 (primario/secundario)
55     int posClave; //indice del campo del registro que forma la clave.
56     bool modif; //Flag para indicar si el indice ha sido modificado.
57
58 public:
59     //Constructor de la clase INDICEND, hereda de ESFicheroBinario anaydiendo nuevas funcionalidades
60     INDICEND(char* nombreFichero,int tamMax,modoApertura mA,tipoIndice tI,int posClave);
61     ~INDICEND();
62     //Metodo que creara el indice del archivo de bloques
63     int creaIndiceNoDenso(FileBloques&);
64     //Realiza un listado de los registros del indice
65     int listarIndiceNoDenso();
66     //Realiza un listado ordenado de los registros del archivo de bloques
67     int listarOrdenadoIPNoDenso(FileBloques& fileBloques);
68     //Busca un registro a partir de la clave primaria devolviendo o la posicion de indice o la del bloque
69     long busquedaBinariaIPNoDenso(char *clave,bool posEnIndice);
70     //Inserta nuevos registros en el archivo de bloques desde ficheroTxt
71     int InsertarNuevosRegistros(fstream &ficheroTxt,FileBloques &fileBloques);
72     //Métodos para leer un registro del indice.
73     long LeerRegistroIP(registroIP &indicePrimario);
74     long LeerRegistroIP(registroIP &indicePrimario,long pos);
75 };

```

Esta clase representa al índice no denso, descende de **ESFicheroBinario** que fue creado en la primera práctica, y añadimos las mismas propiedades que se uso en la práctica 2.

Métodos INDICEND

Generación Archivo de Índices

```

31 int INDICEND::creaIndiceNoDenso(FileBloques& fileBloques)
32 {
33     Registro reg;
34     BloqueIND b(fileBloques.cabecera->tamMaxReg); //bloque auxiliar
35     registroIP regIP;
36     int numIndices=0;
37     long pos;
38     while (!fileBloques.fichero.eof())
39     {
40         pos=fileBloques.LeerBloque(b); //leo bloque
41         if (pos!=-1)
42         {
43             b.getUltimoReg(reg); //obtengo reg en last post
44             regIP.indice.posicionReg=pos;
45             strcpy(regIP.indice.clavePrim,reg.Bridge.Id); //inicializo registro IP
46             numIndices++;
47             regIP.EstablecerCadenaRegistro(ESRegBinario->GetCadenaRegistro());
48             if (EscribirCadenaRegistro()==-1) return -1;
49             cout<<regIP<<endl;
50         }
51     }
52     cabecera->numRegs=numIndices;
53     return numIndices;
54 }

```

Este método se encarga de la creación del **archivo de índices** a partir del archivo de bloques ya creado. El método ira leyendo cada bloque [38,51]. Si no hay problemas obtengo el último registro del bloque [43] e inicializo las propiedades del registro de **índice primario** [44,45], posteriormente lo escribo [48] y finalmente actualizo el **numero de registros insertados**, devolviendo este número [52,53].

Listado Ordenado de Archivo de Bloques

```

65  int INDICEND::listarOrdenadoIPNoDense(FileBloques& fileBloques)
66  {
67      registroIP regIP;
68      long pos;
69      BloqueIND b(fileBloques.cabecera->tamMaxReg);
70      int i;
71      for (i=0;i<cabecera->numRegs;)
72      {
73          if (LeerRegistroIP(regIP)==-1)break;
74          if (fileBloques.LeerBloque(b,regIP.indice.posicionReg)!=-1)
75          {
76              b.mostrar();
77              i++;
78              while ((pos=b.bloque.posSig)!=-1)
79              {
80                  fileBloques.LeerBloque(b,pos);
81                  b.mostrar();//voy mostrando bloques enlazados a b
82                  i++;
83              }
84          }
85          else return -1;
86      }
87      return i;//marcara el numero de bloques mostrados
88  }

```

Este método se encarga de realizar la lectura ordenada del **archivo de bloques**, para ello va leyendo los registros del archivo de **índice no denso** [73] y posteriormente leyendo el bloque usando la posición asociada al índice [74]. Para cada bloque lo muestro y además realizo el recorrido por la lista enlazada en la **zona de overflow**, mostrando los bloques que vamos consiguiendo [78,83]. Finalmente devuelvo la variable i que valdrá el número de bloques mostrados.

Lectura registro de indice

```

178  long INDICEND::LeerRegistroIP(registroIP &regIndicel,long pos)
179  {
180      fichero.seekg(pos,ios::beg);
181      return LeerRegistroIP(regIndicel);
182  }
183  long INDICEND::LeerRegistroIP(registroIP &regIndicel)
184  {
185      long pos=LeerCadenaRegistro();
186      if (pos==-1)return -1;
187      regIndicel.ObtenerCadenaRegistro(ESRegBinario->GetCadenaRegistro());
188      return pos;
189  }

```

Son los mismo que se usaron en la practica 2, adaptados para la clase INDICEND. Uno sin especificación de posición y otro en la que se especifica en el que se llama al anterior.

Búsqueda Binaria en Archivo Índice

```

89 long INDICEND::busquedaBinariaIPNoDense(char *clave, bool posEnIndice){
90     int inf=0;
91     int sup=(cabecera->numRegs)-1;
92     int centro=0;
93     long pos;
94     registroIP regIP;
95     registroIP auxIP;
96     while (inf<=sup){
97         centro=(inf+sup)/2;
98         if (LeerRegistroIP(regIP,centro*cabecera->tamMaxReg+tamCabecera)==-1)return -1;
99         if (regIP>=clave)//si es mayor o igual que clave puede ser el correcto.
100         {
101             centro--;
102             if (LeerRegistroIP(auxIP,centro*cabecera->tamMaxReg+tamCabecera)==-1){
103                 pos=(centro+1)*cabecera->tamMaxReg+tamCabecera;
104                 break;//si llega aquí será porque es el primer bloque
105             }
106             if (auxIP<clave)//comprobamos si el anterior a regIP es menor que la clave
107             {
108                 //si es así lo hemos encontrado actualizamos pos
109                 centro++;
110                 pos=centro*cabecera->tamMaxReg+tamCabecera;
111                 break;
112             }
113             sup=centro;//no lo hemos encontrado actualizamos límites
114         }
115         else if (regIP<clave){//realizamos las mismas comprobaciones pero al revés
116             centro++;
117             if (LeerRegistroIP(regIP,centro*cabecera->tamMaxReg+tamCabecera)==-1){
118                 pos=(centro-1)*cabecera->tamMaxReg+tamCabecera;
119                 break;//si llega aquí será porque es el último bloque
120             }
121             if (regIP>=clave){
122                 pos=centro*cabecera->tamMaxReg+tamCabecera;
123                 break;
124             }
125             inf=centro;//no lo hemos encontrado actualizamos límites
126         }
127     }
128     fichero.clear();//limpiamos flags
129     //devolvemos la posición del registro en índice o la posición del registro real de datos
130     return posEnIndice? pos:regIP.indice.posicionReg;

```

Con este método conseguimos saber donde se encuentra un registro con un determinado valor de **clave primaria**. Es una **búsqueda binaria** común pero ya que se trata de un **índice no denso** hay a que añadirle algunas peculiaridades. En la variable **pos** guardaremos la posición del registro de índice primario correcto y en **regIP** el propio registro. Cuando leemos un registro del índice [98], comprobamos si este es mayor o igual que la clave buscada [99], si se cumple esto, leemos el anterior [102] y si se cumple que la clave es mayor que la del registro leído **auxIP** [106], entonces **regIP** es el que apunta al bloque correcto. Si el bloque que le corresponde es el primero entrara en el **if** de la línea [103] porque dará un error al leer. Si no se cumple ninguna de las anteriores llegara a [112] en la que restringimos la búsqueda a la parte inferior. En las líneas [114,125] se hace lo mismo pero al revés. Finalmente en [129] devolvemos o la posición del registro o la posición del bloque, en función del parámetro **posEnIndice**.

Clase FileBloques

```

77  class FileBloques:public ESFicheroBinario
78  {
79  private:
80      int factorCarga;//factor de carga del bloque
81      int tamMaxReg; //tamanyo maximo de registros de cada bloque
82      //insertara ordenado un registro en el bloque b
83      char* insertarOrdenado(Registro& reg,BloqueIND& b);
84      //Obtiene en blast ultimo bloque de la lista de b y devuelve su posicion
85      long getUltimoLista(BloqueIND& b,BloqueIND& blast);
86      friend class INDICEND;
87  public:
88      //Constructor de FileBloques, tamMax sera el tamanyo de bloque y tmR el de registro
89      FileBloques(char* nombreFichero,int tamMax,modoApertura mA,int fc,int tmR);
90      ~FileBloques();
91      //Metodo que creara el archivo de bloques con factorCarga registros en cada bloque
92      int creaArchivoBloques(ESFicheroBinario&,INDICE&);
93      //Realiza un listado de los bloques del archivo
94      int listarBloques();
95      //Lee un bloque del archivo binario y lo guarda en un STRUCT.
96      long LeerBloque(BloqueIND &b);
97      long LeerBloque(BloqueIND &b, long pos);
98      //Inserta un registro en el archivo de bloques de pos
99      char* InsertarRegistro(Registro& reg,long pos);
100     //Inserta un registro en el archivo de bloques enlazando b
101     char* InsertarEnlazarEnzonaOverflow(Registro& reg,BloqueIND &b,long posBloque);
102     //Realiza la busqueda de valor en los registros de los bloques enlazados a b
103     int buscarRegZonaOverflow(BloqueIND& b,char* valor);
104 }

```

Esta clase modela al archivo de bloques, desciende también de **ESFicheroBinario**, y le añadimos la propiedad factor de Carga (será el factor de Blocaje) y el tamaño máximo de registro, la hacemos amiga de INDICEND para que esta pueda acceder a sus propiedades. Explicaremos más adelante alguno de sus métodos más importantes.

Métodos FileBloques

Generación Archivo de Bloques

```

196  int FileBloques::creaArchivoBloques(ESFicheroBinario& ESfile,INDICE& iP)
197  {
198      registroIP regIP;
199      Registro reg;
200      BloqueIND b(cabecera->tamMaxReg);//en cabecera guardamos el tamanyo de bloque
201      int numRegsInsert=0;
202      for (int i=0;i<iP.cabecera->numRegs;i++)
203      {
204          //leemos tantos registros seguidos como factor de carga
205          for (int j=0;j<factorCarga;j++)
206          {
207              if (iP.LeerRegistroIP(regIP)==-1)break;
208              if (ESfile.LeerRegistro(reg,regIP.indice.posicionReg)==-1)return -1;
209              b.addReg(reg);//anyadiendo el registro al bloque
210              numRegsInsert++;
211          }
212          //escribimos el bloque
213          if (b.EstablecerCadenaBloque(ESRegBinario->GetCadenaRegistro())==-1)break;
214          if (EscribirCadenaRegistro()==-1)return -1;
215          b.clear();
216      }
217      cabecera->numRegs=numRegsInsert;//guardo el numRegs en cabecera
218      return numRegsInsert; //para obtener bloques dividido por factorCarga
219  }

```


Este método será el encargado de realizar la construcción del **archivo de bloques**, para ello nos servimos del fichero de índice primario y del archivo de datos original, declaramos los registros y el bloque que usaremos en las líneas [198,200]. Leeremos tantos registros como nos indique el **índice primario**, y dentro realizamos otro bucle que se repetirá tantas veces como registros queramos tener en cada bloque .En cada iteración leemos del fichero de índices y luego del fichero de datos y añadimos el registro al bloque [205,211]. Después escribimos el bloque completo en el archivo y lo restreamos [213,215]. Finalmente establecemos el número de registros insertado y lo devolvemos. Destacar que devolvemos el número de registros porque si nos hace falta el número de bloques bastaría con dividir este número por el **factor de blocaje**.

BusquedaEnZonaOverflow

```

351  int FileBloques::buscarRegZonaOverFlown(BloqueIND& b,char* valor)
352  {
353      BloqueIND aux(cabecera->tamMaxReg);
354      aux=b;          //salvo bloque en aux
355      long pos;
356      Registro reg;
357      bool encontrado=false;
358      vector<string> regs;
359      while ((pos=aux.bloque.posSig)!=-1&&!encontrado)
360      { //avanzo por la lista hasta que encuentre uno sin siguien
361          if (LeerBloque(aux,pos)==-1)return -1;
362          if (aux.buscar(valor))encontrado=true;
363      }
364      return encontrado;
365  }

```

Este método realiza la búsqueda en la zona de **overflow** de un registro con un determinado valor de clave partiendo de un bloque inicial. Se va siguiendo la lista enlazada que establece la propiedad **posSig** del bloque y cada vez que tenemos uno valido hacemos uso del método buscar del bloque. Devolvemos encontrado.

Constructores

El constructor de las clases que heredan de **ESFicheroBinario** simplemente lo llaman siguiendo la sintaxis adecuada y luego inicializan sus propias propiedades, el de la case **BloqueIND** reserva espacio para la cadena de registros e inicializa nRegs a 0 y el siguiente a -1.

```

20  //METODOS INDICEND
21  INDICEND::INDICEND(char* nombreFichero,int tamMax,modoApertura mA,tipoIndice tipInd,int pos)
22      :ESFicheroBinario(nombreFichero,tamMax,mA)
23  {
24      tI=tipInd;
25      posClave=pos;
26      modif=false;
27  }
191  FileBloques::FileBloques(char* nombreFichero,int tamMax,modoApertura mA,int fc,int tmR)
192      :ESFicheroBinario(nombreFichero,tamMax,mA)
193  {
194      factorCarga=fc;
195      tamMaxReg=tmR;
196  }
385  BloqueIND::BloqueIND(int tamMaxBloque,int sig)
386  {
387      bloque.cadRegs=new char[tamMaxBloque];
388      bloque.posSig=sig;
389      bloque.nRegs=0;
390  }

```


Operación de Inserción

La operación de inserción interviene todas las clases creadas, empezaremos por el método base que pertenece a la clase INDICEND.

```

138  int INDICEND::InsertarNuevosRegistros(fstream &ficheroTxt, FileBloques &fileBloques)
139  {
140      registroIP regIP;
141      Registro registro;
142      char *cadenaRegistroTxt=new char[200];
143      int numRegsInsertados=0;
144      long posBloque,posIndice;
145      char* claveIndice;
146      while (!ficheroTxt.eof())
147      {
148          //leemos la cadena desde el txt y convertimos en registro
149          fileBloques.LimpiarCadena(cadenaRegistroTxt);
150          if (fileBloques.LeerRegistroTxt(ficheroTxt,cadenaRegistroTxt)==-1)break;
151          fileBloques.FormatearCadenaRegistro(cadenaRegistroTxt);
152          cout<<fileBloques.ESRegBinario->GetCadenaRegistro()<<endl;
153          registro.ObtenerCadenaRegistro(fileBloques.ESRegBinario->GetCadenaRegistro());
154          //hallamos posiciones del bloque y del indice correspondientes
155          posBloque=busquedaBinariaIPNoDenso(registro.Bridge.Id,false);
156          posIndice=busquedaBinariaIPNoDenso(registro.Bridge.Id,true);
157          //insertamos registro en bloque correspondiente
158          claveIndice=fileBloques.InsertarRegistro(registro,posBloque);
159          fileBloques.cabecera->numRegs++;
160          numRegsInsertados++;
161          //actualizamos si es necesario ultima clave de bloque
162          if (claveIndice)
163          {
164              if (LeerRegistroIP(regIP,posIndice)==-1)return -1;
165              if (strcmp(regIP.indice.clavePrim,claveIndice)!=0)
166              {
167                  strcpy(regIP.indice.clavePrim,claveIndice);
168                  fichero.seekp(posIndice,ios::beg);
169                  regIP.EstablecerCadenaRegistro(ESRegBinario->GetCadenaRegistro());
170                  if (EscribirCadenaRegistro()==-1)return -1;
171              }
172              delete[] claveIndice;
173          }
174      }
175      delete[] cadenaRegistroTxt;
176      return numRegsInsertados;
177  }

```

Este método será el que se llamara desde el **programa principal**, usara el archivo de bloques y el fichero de texto con los nuevos registros. Ira leyendo en bucle cada cadena del fichero [146,174], convirtiéndola a registro [153] y hallando la posición del bloque y del registro de índice que le corresponde [155,156]. Tras esto usamos el método de la clase **FileBloques** para insertar un registro y actualizamos la **clave del índice** [162,173] en el caso de que sea necesario. Finalmente devolvemos el **número de registros insertados**.

Insercion de Registro

Este método es la base de la inserción de un registro en el fichero de bloques. Primero leemos el bloque [256] y comprobamos si está lleno [257]. Si no lo está **insertamos ordenado** el registro en el bloque [258] y lo escribimos en el archivo [259,261] devolviendo la última clave [262]. Si está lleno obtenemos el último bloque de la lista [265,268] y el último registro [269]. De nuevo comprobamos si está lleno [270], si no, actuamos como antes [271,275], si está lleno, comprobamos si el **nuevo** registro es menor que el **último** [278], si es menor eliminamos el **último** [279], insertamos el **nuevo** [280] e insertamos el **último** en zona de **overflow** [281], si es mayor el **nuevo** registro [282] lo insertamos en la zona de **overflow** [283], devolviendo en ambos casos la ultima clave.

```

251 char* FileBloques::InsertarRegistro(Registro &reg,long posBloque) {
252     BloqueIND b(cabecera->tamMaxReg);
253     Registro last;
254     char* clave;
255     long pos;
256     if (LeerBloque(b,posBloque)==-1)return NULL;
257     if (b.bloque.nRegs<factorCarga) { //insercion en zona maestra si hay espacio
258         clave=insertarOrdenado(reg,b); //inserto ordenado
259         fichero.seekp(posBloque,ios::beg);
260         if (b.EstablecerCadenaBloque(ESRegBinario->GetCadenaRegistro())==-1)return NULL;
261         if ((pos=EscribirCadenaRegistro())==-1)return NULL;
262         return clave;
263     } else { //obtengo ultimo bloque de la lista de b
264         BloqueIND blast(cabecera->tamMaxReg);
265         if (b.bloque.posSig==1) {
266             pos=posBloque;
267             blast=b;
268         } else pos = getUltimoLista(b,blast);
269         blast.getUltimoReg(last); //obtengo ultimo reg del bloque
270         if (blast.bloque.nRegs<factorCarga) { //insercion directa si hay espacio en el ultimo
271             clave=insertarOrdenado(reg,blast);
272             fichero.seekp(pos,ios::beg);
273             if (blast.EstablecerCadenaBloque(ESRegBinario->GetCadenaRegistro())==-1)return NULL;
274             if ((pos=EscribirCadenaRegistro())==-1)return NULL;
275             return clave;
276         } else {
277             //si es menor que el ultimo
278             if (reg<last) {
279                 blast.eliminarUltimoReg(factorCarga); //elimino el ultimo
280                 insertarOrdenado(reg,blast); //inserto el nuevo
281                 return InsertaryEnlazarEnzonaOverflow(last,blast,pos); //e inserto el ultimo en zona overflow
282             } else if (reg>last) //si es mayor que el ultimo inserto el nuevo en zona overflow
283                 return InsertaryEnlazarEnzonaOverflow(reg,blast,pos);
284         }
285     }
286     return NULL;
287 }

```

Inserción de Registro en Overflow

```

258 char* FileBloques::InsertaryEnlazarEnzonaOverflow(Registro& reg,BloqueIND &b,long posBloque)
259 {
260     BloqueIND aux(cabecera->tamMaxReg); //salvo bloque en aux
261     aux=b;
262     char* claveIndice;
263     long pos; //avanzo por la lista hasta que encuentre uno sin siguiente
264     while ((pos=aux.bloque.posSig)!=-1)
265     {
266         if (LeerBloque(aux,pos)==-1)return NULL;
267         if (aux.bloque.nRegs<factorCarga)break; //o uno que no este lleno
268     }
269     if (pos==-1) //si no tiene siguiente y esta lleno
270     {
271         BloqueIND newB(cabecera->tamMaxReg); //creo uno nuevo y anyado el nuevo registro
272         newB.addReg(reg);
273         fichero.seekp(0,ios::end);
274         if (newB.EstablecerCadenaBloque(ESRegBinario->GetCadenaRegistro())==-1)return NULL;
275         if ((pos=EscribirCadenaRegistro())==-1)return NULL;
276         aux.bloque.posSig=pos; //actualizando enlaces
277         fichero.seekp(posBloque,ios::beg);
278         if (aux.EstablecerCadenaBloque(ESRegBinario->GetCadenaRegistro())==-1)return NULL;
279         if ((pos=EscribirCadenaRegistro())==-1)return NULL;
280         claveIndice=new char[10];
281         strcpy(claveIndice,reg.Bridge.Id); //guardo clave ultima por si es necesario modificar el indice
282     }
283     else //si hay espacio en el bloque
284     {
285         //inserto en el bloque y recupero clave indice ultima
286         claveIndice=insertarOrdenado(reg,aux);
287         fichero.seekp(pos,ios::beg);
288         if (aux.EstablecerCadenaBloque(ESRegBinario->GetCadenaRegistro())==-1)return NULL;
289         if ((pos=EscribirCadenaRegistro())==-1)return NULL;
290     }
291     return claveIndice;

```

Este método se encarga de introducir el **nuevo** registro en la zona de **overflow** enlazando los bloques. Primero avanzamos por la **lista de bloques** buscando uno que no tenga siguiente o que no esté lleno [264,268]. Si está lleno y no tiene siguiente creamos un bloque nuevo añadiendo el **nuevo** registro y escribiendo este bloque en el archivo [269,275], a continuación **actualizamos el enlace del bloque**

escribiéndolo en el archivo y actualizando la clave índice [270,281]. Si no está lleno **insertamos ordenado** el registro y escribimos el bloque [283,290].

Inserción Ordenada

```

235 char* FileBloques::insertarOrdenado(Registro& reg,BloqueIND& b)
236 {
237     int i=b.bloque.nRegs;
238     vector<Registro> regs;
239     b.toVector(regs); //convierto bloque en un vector para facilitar tarea de ordenacion
240     bool meter=false;
241     char* claveIndice=new char[10]; //guardaremos ultimaClave
242     while (i>0&&!meter)
243     {
244         if (reg<regs[i-1]) //si el nuevo es menor que i-1
245         {
246             //muevo i-1 a i
247             copy ( regs.begin()+i-1, regs.begin()+i, regs.begin()+i );
248             i--;
249         }
250         else meter=true;
251     }
252     vector<Registro>::iterator it;
253     regs.insert(regs.begin()+i,reg); //inserto en pos i
254     b.bloque.cadRegs[0]='\0';
255     b.bloque.nRegs=0; //reseteo bloque y anyado los registros ordenados
256     for (unsigned int i=0;i<regs.size();i++)b.addReg(regs[i]);
257     strcpy(claveIndice,regs[regs.size()-1].Bridge.Id);
258     return claveIndice;
259 }

```

Este método **inserta ordenado** un registro en el bloque. Para ello modelamos el bloque como un vector mediante el método de la clase **BloqueIND tovector()** [239], y realizamos el algoritmo de la **inserción ordenada** [242,251]. Tras realizar los desplazamientos correspondiente e insertar el registro en el vector, **reseteamos el bloque** [252,253] y lo creamos de nuevo añadiendo los nuevos registros a partir del vector ordenado [254]. Finalmente devolvemos la **última clave del bloque** [255,256].

Programa Principal creaNODENSO

La mayoría de las peculiaridades del programa se encuentran explicadas en el propio código, dejo aquí una imagen del programa principal como muestra. Destacar que antes de nada el programa pide el **factor de blocaje** que se usará los bloques para que se mantenga constante durante toda la ejecución.

```

----- PRACTICA 2b OGA -----
----- JOSE ANGEL GARCIA FERNANDEZ -----
--> 1 - Crear Archivo Bloques
--> 2 - Crear Archivo Indice Primario No Denso
--> 3 - Listar Archivo Bloques
--> 4 - Listar Indice Primario No Denso
--> 5 - Listar Ordenado Archivo Bloques
--> 6 - Busqueda en Archivo por Clave Primaria
--> 7 - Insertar registros en archivo de datos
--> 0 - Salir
-----
-->

```


Material Adicional

Información encontrada en:

- Transparencias de clase.
- Guión Práctica
- Libro de Folk
- IDE Codeblocks (para descargarlo pulse en el tercer enlace)
- Webs
 - <http://www.cplusplus.com/>
 - <http://www.conclase.net/>
 - <http://www.codeblocks.org/downloads/5>

Dificultades encontradas

Para hacer el método **toVector()** tuve una cantidad de problemas enorme relacionadas con el tratamiento de las cadenas c, al final opte por usar **string** que no me dio ningún problema.

Los problemas típicos de **flags** de los ficheros han vuelto a dar problemas, ya que algunas veces se quedaban activadas aunque cerrábamos el archivo, con lo que teníamos que hacer un **.clear()** para resetearlas.

El método de **busquedaBinaria()** también ha sido bastante complicado, ya que lo he tenido que modificar bastante desde el de la practica2.

Relativas a la **inserción** he tenido también muchos problemas, porque en un principio pensé en juntarlo todo en un método pero conforme iba creciendo la complejidad tuve que ir dividiendo el algoritmo para que fuera más fácil de implementar.

Notas

Para abrir los proyectos use el **IDE Codeblocks** citado anteriormente.

- Archivos:
 - Bridges.dat
 - Archivo de registros apilado
 - BridgesB.dat
 - Archivo de registros en bloques
 - bridgesORIGINAL.dat
 - Archivo de registros original para uso en caso de reseteo
 - IndiceP.dat
 - Archivo de índice primario
 - IndiceND.dat
 - Archivo de índices no denso