Tecnicatura Superior en Análisis, Desarrollo y Programación de Aplicaciones

Programación I

"Trabajando con la Librería idxg"
Versión 1.2 Abril 2018

Mg. Guillermo R. Cherencio

INDICE

Introducción

La librería idxg es un desarrollo hecho dentro de la asignatura Programación I de la Tecnicatura Superior en Análisis, Desarrollo y Programación de Aplicaciones, para facilitar al alumno el manejo de archivos indexados. La misma puede ser utilizada en la implementación de los trabajos finales de la asignatura. Están disponibles los programas fuentes de la librería y en la carpeta /lib del proyecto geany idxg está disponible en forma binaria para su uso y distribución en cualquier proyecto. Este documento forma parte de la documentación de la librería que se encuentra en /lib.

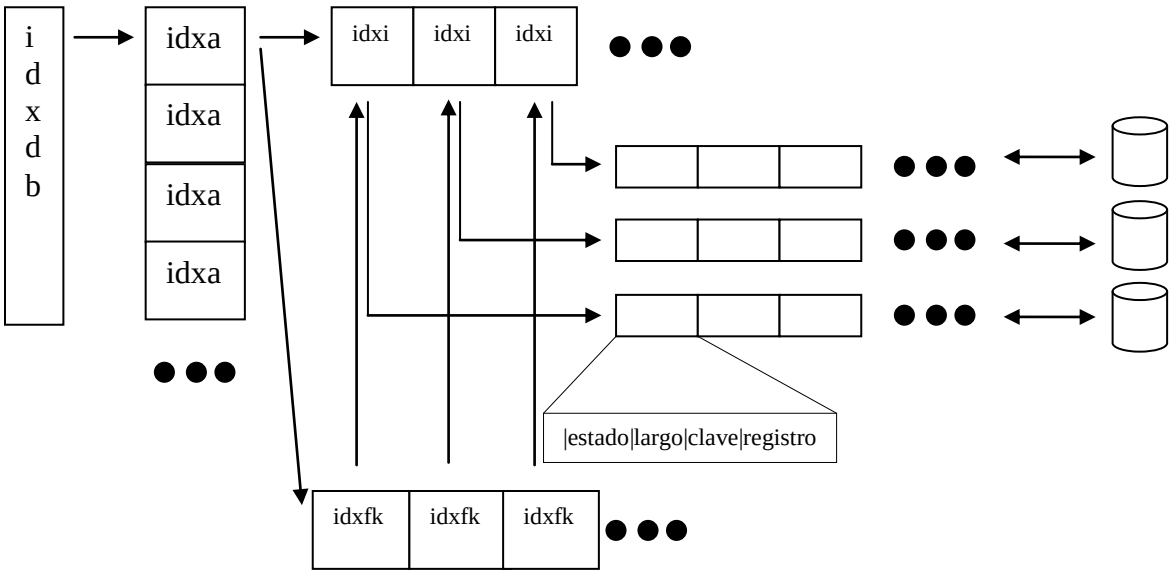
Diseño

Esta librería fue diseñada teniendo en cuenta los siguientes requerimientos:

- Facilidad de uso (uso de estructuras para simplificar API)
- Ocultar al usuario la complejidad del manejo de índices y archivos
- Uso de índices densos en memoria
- Diseño simple aunque no sea el más optimo
- Solo el archivo de datos no se encuentra en memoria
- Permitir más de un índice por archivo
- Permitir el trabajo con un conjunto de archivos
- Los índices pueden ser primarios (no admiten duplicación de claves) o secundarios (una misma clave que apunta a múltiples registros de datos)
- Ordenamiento de índices mediante qsort() y búsqueda mediante bsearch()
- Cuando se actualiza el archivo de datos, los índices se actualizan automáticamente
- Se utiliza un byte de estado (siempre es el primer byte) para indicar registro borrado u ocupado (tanto en índice como en archivo de datos)
- Los índices se cargan desde archivo binario a memoria, un archivo por índice
- Los archivos índices en disco se mantienen siempre con registros no borrados y ordenados físicamente por el valor de clave
- Permitir re-generar un índice, volver a indexar
- Proyecto para generar librería a partir de fuentes y proyecto de ejemplo de uso de librería
- Permitir recorrer índice en sentido ascendente o descendente

- Librería genérica usable sobre cualquier archivo de datos, el único requisito es el byte de estado al comienzo de cada registro
- Permitir volcar el contenido de cualquier archivo índice en pantalla o enviarla a un stream de datos
- Se debería generar otra librería el manejo de consola (por ahora está incluido en la misma librería)
- Se definieron distintos tipos de datos para el manejo de claves, de modo tal, de usar funciones provistas por la librería para el ordenamiento de claves, evitando esta tarea al programador
- Permitir activar o desactivar la emisión de mensajes de trace (traza, seguimiento, ayuda) sobre cada índice para facilitar la búsqueda de errores

Teniendo en cuenta los requerimientos, se diseñó la librería utilizando sólo tres estructuras en memoria. Una estructura define al concepto de "base de datos" (bd) que aquí lo usaremos para referirnos a una colección de archivos relacionados; a esta estructura la llamamos idxdb. Una estructura define al archivo de datos (idxa) y otra estructura define a un índice (idxi). Todas las estructuras se implementan como tipos de datos. La estructura idxdb tiene un puntero (idxdb.pidxa) a un arreglo de N elementos idxa. La estructura idxa tiene un puntero (idxa.pixdi) a un arreglo de N elementos idxi. Cada elemento del arreglo idxi, representa a un índice (ya sea primario o secundario) asociado con el archivo de datos representado por idxa:



Cada elemento del arreglo idxi, tiene –a su vez- un puntero (idxi.base) a un arreglo en memoria que contiene un elemento por cada valor de clave que tiene el índice. Sobre este arreglo se aplica qsort() y bsearch() y esta área de memoria es cargada desde el archivo índice asociado o bien guardada en dicho archivo cuando se deja de utilizar el índice y éste ha sido modificado.

Adicionalmente la estructura idxa tiene un puntero (idxa.pfk) a un arreglo de N elementos idxfk que representan N claves extranjeras. Cada clave extranjera referencia a un índice primario (que no admite repeticiones de valores de clave) previamente definido. La librería idxg no admitirá violaciones de claves extranjeras, emitirá error y el registro no será insertado o actualizado en el archivo de datos.

La estructura de tipo idxdb tiene los siguientes campos:

Campo	Tipo de dato	Descripción
archivo_xml	char[256]	Nombre completo del archivo xml[1] de configuración de la bd que se encuentra en en disco
archivos	char *	Puntero a cadena de caracteres en memoria que representa el contenido del tag <archivos> del archivo xml de configuración. Está disponible en memoria para facilitar el parsing/interpretación del mismo sin necesidad de ir a cargarlo a memoria en forma reiterativa.
nidxa	int	Cantidad de archivos que forman esta bd. Cantidad de elementos que tiene el arreglo de tipo idxa apuntado por pidxa
pidxa	idxa *	Puntero al arreglo de tipo idxa de nidxa elementos, que representa el acceso a cada uno de los archivos de esta bd.

La estructura de tipo idxa tiene los siguientes campos:

Campo	Tipo de dato	Descripción
archivo	char[256]	Nombre completo del archivo de datos binario en disco
tam_registro	int	Longitud/tamaño en bytes de registro, de cada registro de datos de longitud fija.
tam_clave	int	Cantidad de bytes que ocupa el campo clave en el índice.
f	FILE *	Manipulador (handle) del stream de datos asociado con archivo, todas las operaciones de I/O sobre el archivo de datos se hacen a través de este campo.
nsk	int	Contador que mantiene la cuenta de cada uno de los índices asociados con este archivo de datos. Representa la cantidad de elementos del arreglo de tipo idxi apuntado por pidxi.
pidxi	Idxi *	Puntero al arreglo de tipo ixdi de nsk elementos, que representa el acceso a cada uno de los índices de este

[1] Ver sección “Uso de Archivos xml”

		archivo de datos.
nfk	int	Contador que mantiene la cuenta de cada una de las claves extranjeras asociadas con este archivo de datos. Representa la cantidad de elementos del arreglo de tipo idxfk apuntado por pfk.
pfk	idxfk *	Puntero al arreglo de tipo idxfk de nfk elementos, que representa el acceso a cada una de las claves extranjeras de este archivo de datos.

La estructura de cada elemento de tipo idxi es la siguiente:

Campo	Tipo de dato	Descripción
archivo	char[256]	Nombre completo del archivo índice binario en disco
tam_registro	int	Longitud/tamaño de registro de cada elemento del arreglo que contiene los valores de clave descripto en el cuadro anterior.
tam_clave	int	Cantidad de bytes que ocupa el campo clave en el índice.
tipo	unsigned char	Tipo de índice IDXG_PK (0, clave primaria) o bien IDXG_SK (1, clave secundaria).
pos_clave	int	Posición de la clave dentro del registro de datos, en donde se encuentra la clave (offset, desplazamiento de base cero).
off_clave	int	Posición de la clave dentro del registro índice.
off_registro	int	Posición del número de registro del registro índice.
base	void *	Puntero al índice en memoria.
posicion	void *	Posición actual en el índice, apunta a un elemento dentro del arreglo apuntado por base.
n	long	Cantidad de claves en el índice apuntado por base.
comparo	int (*)(const void *, const void *)	Función de comparación utilizada por qsort() y bsearch().
modificado	char	Flag, bandera, que indica que si el contenido del índice fue modificado o no.
trace	char	Flag, bandera, que indica que si se deben emitir mensajes de ayuda/trace o no.
tipo_dato	char	Tipo de dato de la clave del índice, determina las funciones de ordenamiento y búsqueda.
tam_output	int	Cantidad de bytes máximo para mostrar clave en formato string

La estructura de cada elemento de tipo idxfk es la siguiente:

Campo	Tipo de dato	Descripción
archivo	char[256]	Nombre lógico de archivo de datos que referencia la clave extranjera
nombre	char[256]	Nombre lógico de índice definido en el archivo de datos referenciado por la clave extranjera
posicion	int	Posición de la clave extranjera dentro del archivo de datos
referencia	Idxi *	Puntero a entrada índice que referencia esta clave extranjera. El índice apuntado no admite repeticiones.

Las estructura de cada elemento del arreglo que contiene cada valor clave, es la siguiente:

Campo	Largo en bytes	Descripción
estado	1	Indica si esta clave esta borrada o no
largo	1	Indica el largo de la clave en bytes. Dato redundante que facilita la implementación de la función de callback requerida por qsort() y bsearch().
clave	largo bytes	Valor de clave que tiene largo bytes de longitud.
registro	sizeof(long)	Puntero al archivo de datos, número de registro (0 .. N-1) en donde se encuentra el registro de datos que contiene este valor de clave.

Macros, Constantes, Define's de Librería idxg

Expresión	Tipo	Descripción
IDXG_PK	Define. Valor entero, 0.	Indica que el tipo de índice es PK, índice que no admite repetición de claves.
IDXG_SK	Define. Valor entero, 1.	Indica que el tipo de índice es SK, índice que admite repetición de claves.
IDXG_BORRADO	Define. Valor char, '2'.	Valor que indica entrada en registro índice o registro de datos borrado.
IDXG_NUEVO	Define. Valor char, '1'.	Valor que indica entrada en registro índice o registro de datos NO borrado.

Funciones Librería idxg

A continuación se documentan –en orden alfabético- las funciones públicas que forman parte de API de la librería idxg:

idxg_abrir

Descripción	Función inicial que permite crear una bd en memoria a partir del archivo xml de configuración de la misma. Lee el archivo xml, hace su parsing/interpretación, crea en memoria todos los archivos e índices, llama a idxg_leer_xml(), devuelve puntero a estructura bd o bien NULL si hubo error
Prototipo	idxdb *idxg_abrir(char *archivo_xml)

idxg_anterior

Descripción	Me paro y devuelvo la anterior entrada del índice que no esté borrado a partir de la posición actual. Actualiza el campo posición. Devuelve el registro de dato correspondiente a la posición del anterior elemento índice en memoria no borrado o bien NULL en caso de que no haya índice en memoria o bien estén borrados todos sus elementos o bien se haya llegado al final del índice
Prototipo	void *idxg_anterior(idxa *p,int idíndice)

idxg_archivo

Descripción	Devuelve puntero a archivo a partir del número del mismo (0..N-1) o NULL en caso de idindice incorrecto
Prototipo	idxa *idxg_archivo(idxdb *db,int id)

idxg_archivo2

Descripción	Idem idxg_archivo() pero esta función ubica al archivo a través de su nombre en vez de su id
Prototipo	idxa *idxg_archivo2(idxdb *db,char *archivo)

idxg_borrar

Descripción	Borra el registro de datos, actualiza todos los índices, hace baja.
Prototipo	void idxg_borrar(idxa *p,void *data)

idxg_buscar

Descripción	Devuelve el registro índice encontrado o NULL si no existe o esta borrado (NO liberar esta memoria!)
Prototipo	void *idxg_buscar(idxa *p,int idíndice,void *clave)

idxg_buscard

Descripción	Devuelve el registro de datos encontrado, devuelve NULL si no existe (luego liberar esta memoria) o esta borrado
Prototipo	void *idxg_buscard(idxa *p,int idíndice,void *clave)

idxg_buscarr

Descripción	Devuelve el registro índice encontrado o NULL si no existe (NO liberar esta memoria!). Busca por clave y número de registro.
Prototipo	void *idxg_buscarr(idxa *p,int idíndice,void *clave,long registro)

idxg_buscarrd

Descripción	Devuelve el registro de datos encontrado o NULL si no existe (luego liberar esta memoria) busca por clave y número de registro.
Prototipo	void *idxg_buscarrd(idxa *p,int idíndice,void *clave,long registro)

idxg_buscod

Descripción	Idem idxg_buscard() pero buscando por clave primaria.
Prototipo	void *idxg_buscod(idxa *p,void *clave)

idxg_cerrar

Descripción	Cierra una bd, libera memoria de todos los archivos e índices asociados a la misma
Prototipo	void idxg_cerrar(idxdb *db)

idxg_clavestr

Descripción	Devuelve la clave en formato string para ser mostrada en pantalla o enviada a stream de output. La clave en formato string se copia en claveout . claveout debe tener -al menos- i->tam_output bytes para almacenar clave en formato string. Utilizada por funcion idxg_mostrar(), acorde con el tipo de dato de la clave.
Prototipo	void idxg_clavestr(idxa *p,int idíndice,void *clave,char *claveout)

idxg_comparo

Descripción	Función callback utilizada por qsort() y bsearch() acorde con tipo de dato de la clave del índice (también llamada clave de indexación). Función standard de comparación utilizada cuando no se indica tipo o el mismo es desconocido. Si el campo registro correspondiente al primer argumento de la función vale -1L esto indica que sólo se busca por valor clave, sin tener en cuenta el valor del campo registro. Caso contrario, el campo registro también interviene en la comparación de claves, ya sea para ordenamiento o búsqueda.
Prototipo	int idxg_comparo(const void *, const void *)

idxg_comparo_char, idxg_comparo_double, idxg_comparo_int, idxg_comparo_long, idxg_comparo_str

Descripción	Idem idxg_comparo(), adaptadas para cada uno de los tipos de datos de la clave índice.

Prototipos	int idxg_comparo_char(const void *, const void *);
	int idxg_comparo_double(const void *, const void *);
	int idxg_comparo_int(const void *, const void *);
	int idxg_comparo_long(const void *, const void *);
	int idxg_comparo_str(const void *, const void *);

idxg_fin

Descripción	Libera todos los recursos. Elimina todos los índices de memoria. Pone campo nsk en 0. Graba todos los índices que sean necesarios. Cierra archivo de datos. Llamar a esta función antes de terminar de trabajar con la librería.
Prototipo	void idxg_fin(idxa *)

idxg_grabar_xml

Descripción	Genera, crea archivo xml de configuración a partir de los archivos

	actuales en memoria. Función inversa a idxg_leer_xml()
Prototipo	void idxg_grabar_xml(idxa *p, char *archivo_xml)

idxg_guardar

Descripción	Guarda el registro de datos, actualiza todos los índices, hace alta y/o modificación de registros de datos.
Prototipo	void idxg_guardar(idxa *p,void *data)

idxg_id_indice

Descripción	Dado el nombre de un índice (su nombre archivo índice, su ubicación), devuelve su id o -1 si no existe
Prototipo	int idxg_id_indice(idxa *p,char *archivo)

idxg_id_indicepk

Descripción	Devuelve el id del primer índice no secundario que encuentre asociado al archivo p o -1 si no hay ningún índice primario
Prototipo	int idxg_id_indicepk(idxa *p)

idxg_igrabar

Descripción	Graba archivo índice en disco, pone flag de modificado en cero (0). No verifica previamente el flag de modificado, siempre graba; si no existe el archivo índice, crea un archivo nuevo.
Prototipo	void idxg_igrabar(idxa *p,int idíndice)

idxg_indexar

Descripción	Libera el índice actual y vuelve a generarlo a partir del archivo de datos. Graba el índice en disco y lo deja cargado en memoria, posición apunta al primer registro del índice.
Prototipo	void idxg_indexar(idxa *p,int idíndice)

idxg_indice, idxg_indice2

Descripción	Asocia el índice descripto en los argumentos de la función con el archivo de datos indicado. Carga el archivo índice en memoria y queda posicionado en el primer registro. En caso de que no haya registros o no exista el archivo índice, queda posicionado en NULL. Devuelve el id (0..N-1) del índice agregado (de base cero). La función idxg_indice2() es idéntica a idxg_indice() pero es utilizable con claves tipificadas, claves de tipo 'i','l','d', etc cuyo tamaño ya está predeterminado, por lo tanto, esta función no necesita que el usuario índique el tamaño de la clave, lo asume, según el tipo de dato de la clave.

Prototipo	int idxg_índice(idxa *p,char *archivo,int tam_clave_sk,int pos_clave_sk,unsigned char tipo,char tipo_dato) int idxg_indice2(idxa *p,char *archivo,int pos_clave_sk,unsigned char tipo,char tipo_dato)

idxg_indicefk

Descripción	Agrega clave extranjera al archivo p, devuelve id del indice (posición de desplazamiento dentro de idxa.pfk). Nombre archivo se refiere al nombre lógico del archivo (tag nombre dentro de la definición del archivo) y nombre índice se refiere también al nombre lógico del índice dentro del archivo (tag nombre dentro de la definición del índice). El tag referencia dentro de la definición de clave extranjera contiene “nombre de archivo punto nombre de índice”.
Prototipo	int idxg_indicefk(idxdb *db,idxa *p,char *nombre_archivo,char *nombre_indice,int posicion)

idxg_inicio, idxg_inicio2

Descripción	Crea –si es necesario- archivo de datos indicado, si existe lo abre de R/W, el archivo de datos queda abierto. Devuelve puntero a estructura de tipo idxa que representará el archivo de datos, al que luego, se le asociarán los archivos índices que requiera el proceso. La función idxg_inicio2() es idéntica, pero opera sobre una estructura idxa previamente asignada en memoria (esto facilita la implementación del concepto de base de datos)
Prototipo	idxa *idxg_inicio(char *archivo,int tam_registro) void idxg_inicio2(idxa *p,char *archivo,int tam_registro)

idxg_abrir, idxg_leer_xml

Descripción	Lee archivo xml, crea estructura bd, llama a la función idxg_inicio(), idxg_indice(), idxg_indice2(), etc. todas las funciones que correspondan para configurar arreglo de estructura idxa y devuelve puntero idxdb. Esto facilita el uso de la librería a partir de especificar archivos e índices dentro de archivo xml para luego utilizarlo. Configura idxa, carga índices en el mismo orden que están en el archivo xml, configura idxdb.archivo_xml con archivo_xml indicado como argumento
Prototipo	idxdb *idxg_leer_xml(char *archivo_xml) idxdb *idxg_abrir(char *archivo_xml)

idxg_grabar_xml

Descripción	Genera, crea archivo xml de configuración de bd a partir de la información actual en memoria
Prototipo	void idxg_grabar_xml(idxdb *p)

idxg_mostrar

Descripción	Imprime el contenido del índice en stream out, si out es NULL usa stdout. Usa idxg_clavestr() para mostrar clave en formato string.
Prototipo	void idxg_mostrar(idxa *p,int idíndice,FILE *out)

idxg_nueva_bd

Descripción	Crea una nueva bd en memoria, sabiendo de antemano cuantos archivos forman parte de la nueva bd.
Prototipo	idxdb *idxg_nueva_bd(char *archivo_xml,int narchivos)

idxg_primero

Descripción	Se posiciona en índice y devuelve la primer entrada del índice que no esté borrada. Actualiza el campo posición. Devuelve el registro de datos correspondiente a la posición del primer elemento índice en memoria no borrado o bien NULL en caso de que no haya índice en memoria o bien estén borrados todos sus elementos.
Prototipo	void *idxg_primero(idxa *p,int idíndice)

idxg_siguiente

Descripción	Se posiciona en índice y devuelve la siguiente entrada del índice que no esté borrado a partir de la posición actual. Actualiza el campo posición. Devuelve el registro de dato correspondiente a la posición del siguiente elemento índice en memoria no borrado o bien NULL en caso de que no haya índice en memoria o bien estén borrados todos sus elementos o bien se haya llegado al final del índice.
Prototipo	void *idxg_siguiente(idxa *p,int idíndice)

idxg_trace

Descripción	Permite activar (1) o desactivar (0) el trace (mensajes adicionales para encontrar errores) del índice indicado.
Prototipo	void idxg_trace(idxa *p,int idíndice,char trace)

idxg_ultimo

Descripción	Se posiciona en índice y devuelve la última entrada del índice que no esté borrada. Actualiza el campo posición. Devuelve el registro de datos correspondiente a la posición del último elemento índice en memoria no borrado o bien NULL, en caso de que no haya índice en memoria o bien estén borrados todos sus elementos.
Prototipo	void *idxg_ultimo(idxa *p,int idíndice)

idxg_xml_borrar_tag

Descripción	Borra de buffer el contenido del tag nombreTag (lo reemplaza con espacios dentro de buffer). Luego de utilizar esta función, el contenido apuntado por buffer ha sido modificado (sólo en caso de encontrar el tag en la posición esperada). Si desea quitar los espacios y caracteres innecesarios antes y después del valor del tag, deberá utilizar la función idxg_xml_trim().
Prototipo	void idxg_xml_borrar_tag(char *buffer, int posicion, char *nombreTag)

idxg_xml_cargar

Descripción	Lee el archivo xml archivo_xml, lo carga en memoria en buffer terminado en \0 y devuelve puntero a dicho buffer. Liberar esta memoria luego de haber invocado a esta función.
Prototipo	char *idxg_xml_cargar(char *archivo_xml)

idxg_xml_contar_tag

Descripción	Dado el tag nombreTag busca cuantas veces se encuentra ese tag dentro de buffer.
Prototipo	int idxg_xml_contar_tag(char *buffer, char *nombreTag)

idxg_xml_tag

Descripción	Dado el tag nombreTag (el cual puede estar N veces dentro de buffer), la posición del mismo (0 para la primera posición u ocurrencia del tag y así sucesivamente) y el buffer xml donde buscar, devuelve su contenido (al contenido se le ha aplicado idxg_xml_trim() para quitar espacios innecesarios al principio y al final del valor del tag). Liberar esta memoria luego de haber invocado a esta función.
Prototipo	char *idxg_xml_tag(char *buffer, int posicion, char *nombreTag)

Programas de Ejemplo

En la carpeta /lib de distribución de esta librería se incluye el programa testidxg.c que contiene código de ejemplo de uso de esta librería. Dado el archivo de productos y rubros, cuyos registros se representan por las siguientes estructuras, respectivamente:

```
// estructura de registro de un producto
struct pr {
      char estado;
      int codigo;
      char descr[100];
      double precio;
      int rubro;
};
// estructura de registro de un rubro
struct rb {
      char estado;
      int codigo;
      char descr[100];
};
```

En donde estado representa el byte de estado (‘2’ borrado, ‘1’ ocupado, ‘0’ o NULL libre), código representa el código de producto (su clave primaria), descr se refiere a su descripción (sobre los 10 primeros caracteres se crea un índice secundario para ordenamiento/búsqueda de productos por nombre) y rubro también se utiliza para crear un índice secundario sobre el mismo.

Abm sobre productos

```
void abm() {
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"prod.dat");
      idxa *arrb = idxg_archivo2(db,"rubro.dat");
      int idxcodrb = idxg_id_indicepk(arrb);
      if ( ar ) {
            int continuar = 1;
            do {
                  int codigo = ingreso_codigo();
                  if ( codigo == -1 ) { continuar=0;continue; }
                  struct pr *data = (struct pr *)
idxg_buscod(ar,&codigo);
                  if ( data != NULL ) {
                        muestro_producto(data);
                        if ( ingreso_sino("Modifica Producto?:") ) {
                              // cambio
```

```
                              data->estado = '1';
                              ingreso_descr(data->descr);
                              data->precio = ingreso_precio();
                              int rubro_ok=0;
                              do {
                                    data->rubro = ingreso_rubro();
                                    rubro_ok = (
idxg_buscar(arrb,idxcodrb,&data->rubro) != NULL );
                                    if ( !rubro_ok ) {
                                          printf("abm_xml(): rubro %d
no existe!, reintente!\n",data->rubro);
                                          if ( ingreso_sino("Desea
Listar Rubros?:") ) listado_cod_rubro();
                                    }
                              } while(!rubro_ok);
                              idxg_guardar(ar,data);
                        } else if ( ingreso_sino("Borra Producto?:") )
{
                              // baja
                              idxg_borrar(ar,data);
                        }
                        free(data);
                  } else {
                        if ( ingreso_sino("Ingresa Producto?:") ) {
                              // alta
                              struct pr reg;
                              memset(&reg,0,sizeof(struct pr));
                              reg.estado = '1';
                              reg.codigo = codigo;
                              ingreso_descr(reg.descr);
                              reg.precio = ingreso_precio();
                              int rubro_ok=0;
                              do {
                                    reg.rubro = ingreso_rubro();
                                    rubro_ok = (
idxg_buscar(arrb,idxcodrb,&reg.rubro) != NULL );
                                    if ( !rubro_ok ) {
                                          printf("abm_xml(): rubro %d
no existe!, reintente!\n",reg.rubro);
                                          if ( ingreso_sino("Desea
Listar Rubros?:") ) listado_cod_rubro();
                                    }
                              } while(!rubro_ok);
                              idxg_guardar(ar,&reg);
                        }
                  }
                  continuar = ingreso_sino("¿Continua?:");
            } while(continuar);
            idxg_cerrar(db);
            free(db);
      } else {
```

```
            printf("abm_xml(): Error!, no pude crear estructura archivo
en memoria!\n");
      }
}
```

Abm sobre rubros

```
void abm_rubro() {
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"rubro.dat");
      if ( ar ) {
            int continuar = 1;
            do {
                  int codigo = ingreso_rubro();
                  if ( codigo == -1 ) { continuar=0;continue; }
                  struct rb *data = (struct rb *)
idxg_buscod(ar,&codigo);
                  if ( data != NULL ) {
                        muestro_rubro(data);
                        if ( ingreso_sino("Modifica Rubro?:") ) {
                              // cambio
                              data->estado = '1';
                              ingreso_descr(data->descr);
                              idxg_guardar(ar,data);
                        } else if ( ingreso_sino("Borra Rubro?:") ) {
                              // baja
                              idxg_borrar(ar,data);
                        }
                        free(data);
                  } else {
                        if ( ingreso_sino("Ingresa Rubro?:") ) {
                              // alta
                              struct rb reg;
                              memset(&reg,0,sizeof(struct rb));
                              reg.estado = '1';
                              reg.codigo = codigo;
                              ingreso_descr(reg.descr);
                              idxg_guardar(ar,&reg);
                        }
                  }
                  continuar = ingreso_sino("¿Continua?:");
            } while(continuar);
            idxg_cerrar(db);
            free(db);
      } else {
            printf("abm_rubro(): Error!, no pude crear estructura
archivo en memoria!\n");
      }
}
```

Listado por código de producto

```
void listado_codigo() { // listado ordenado por codigo
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"prod.dat");
      int idxcodigo = idxg_id_indicepk(ar);
      struct pr *data = (struct pr *) idxg_primero(ar,idxcodigo);
      if ( data != NULL ) {
            do {
                  muestro_producto(data);
                  free(data);
                  data=idxg_siguiente(ar,idxcodigo);
            } while( data != NULL );
      } else {
            printf("No hay datos a listar!\n");
      }
      idxg_cerrar(db);
      free(db);
}
```

Listado por descripción de producto

```
void listado_nombre() { // listado ordenado por nombre
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"prod.dat");
      int idxdescr = idxg_id_indice(ar,"prod.idx2");
      struct pr *data = (struct pr *) idxg_primero(ar,idxdescr);
      if ( data != NULL ) {
            do {
                  muestro_producto(data);
                  free(data);
                  data=idxg_siguiente(ar,idxdescr);
            } while( data != NULL );
      } else {
            printf("No hay datos a listar!\n");
      }
      idxg_cerrar(db);
      free(db);
}
```

Consulta por rubro de producto

```
void consulta_rubro() { // dado un rubro, muestra todos sus productos
      idxdb *db = idxg_abrir("prod.xml");
```

```
      idxa *ar = idxg_archivo2(db,"prod.dat");
      int idxrubro = idxg_id_indice(ar,"prod.idx1");
      idxg_trace(ar,idxrubro,1);       // habilito mensajes para indice
      int rubro = 0;
      do {
            printf("Rubro (1..N) (-1 para Salir):");
            rubro = ingreso_int();
            if ( rubro != -1 ) {
                  struct pr *data = (struct pr *)
idxg_buscard(ar,idxrubro,&rubro);
                  if ( data != NULL ) {
                        while(data != NULL && data->rubro == rubro) {
                              muestro_producto(data);
                              free(data);
                              data=idxg_siguiente(ar,idxrubro);
                        }
                  } else {
                        printf("No hay productos del rubro
%d\n",rubro);
                  }
            }
      } while ( rubro != -1 );
      idxg_cerrar(db);
      free(db);
}
```

Listado descendente por código de producto

```
void listado_desc_codigo() { // listado descendente ordenado por codigo
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"prod.dat");
      int idxcodigo = idxg_id_indicepk(ar);
      struct pr *data = (struct pr *) idxg_ultimo(ar,idxcodigo);
      if ( data != NULL ) {
            do {
                  muestro_producto(data);
                  free(data);
                  data=idxg_anterior(ar,idxcodigo);
            } while( data != NULL );
      } else {
            printf("No hay datos a listar!\n");
      }
      idxg_cerrar(db);
      free(db);
}
```

Listado descendente por rubro de producto

```
// listado descendente ordenado por rubro
void listado_desc_rubro() {
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"prod.dat");
      int idxrubro = idxg_id_indice(ar,"prod.idx1");
      struct pr *data = (struct pr *) idxg_ultimo(ar,idxrubro);
      if ( data != NULL ) {
            do {
                  muestro_producto(data);
                  free(data);
                  data=idxg_anterior(ar,idxrubro);
            } while( data != NULL );
      } else {
            printf("No hay datos a listar!\n");
      }
      idxg_cerrar(db);
      free(db);
}
```

Listado descendente por descripción de producto

```
// listado descendente ordenado por nombre
void listado_desc_nombre() {
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"prod.dat");
      int idxdescr = idxg_id_indice(ar,"prod.idx2");
      struct pr *data = (struct pr *) idxg_ultimo(ar,idxdescr);
      if ( data != NULL ) {
            do {
                  muestro_producto(data);
                  free(data);
                  data=idxg_anterior(ar,idxdescr);
            } while( data != NULL );
      } else {
            printf("No hay datos a listar!\n");
      }
      idxg_cerrar(db);
      free(db);
}
```

Indexación de producto

```
// permite indexar, volver a generar un índice a partir del archivo de
datos
void indexar() {
      idxdb *db = idxg_abrir("prod.xml");
      idxa *ar = idxg_archivo2(db,"prod.dat");
```

```
      int idx = 0;
      do {
            printf("Id Indice a Indexar (0..%d) (-1 para Salir):",(ar-
>nsk-1));
            idx = ingreso_int();
            if ( idx != -1 ) {
                  if ( idx >= 0 && idx < ar->nsk ) {
                        idxg_trace(ar,idx,1);// habilito mensajes
                        idxg_indexar(ar,idx);
                        idxg_trace(ar,idx,0);// deshabilito mensajes
                  } else printf("indexar(): El Indice ingresado es
incorrecto!\n");
            }
      } while ( idx != -1 );
      idxg_cerrar(db);
      free(db);
}
```

Uso de archivos xml

Los archivos xml son archivos de texto, similares a los archivos html que tienen una forma de uso muy libre por parte del usuario y son especialmente útiles para describir y representar datos. Esto nos permitiría –en nuestro caso- definir una base de datos conformada por un conjunto de archivos (uno o más archivos) y cada uno de los archivos con sus respectivos índices, veamos un ejemplo (el contenido del archivo “prod.xml” que define una bd formada por dos archivos de datos (prod.dat y rubro.dat) y una serie de índices para cada archivo) :

```
<?xml version="1.0" encoding="UTF-8"?>
<archivos>
   <archivo>
      <ubicacion>prod.dat</ubicacion>
      <registro>
         <largo>128</largo>
      </registro>
      <indices>
         <indice>
            <ubicacion>prod.idx0</ubicacion>
            <tipo>0</tipo>
            <clave>
               <largo>4</largo>
               <posicion>4</posicion>
               <tipo>i</tipo>
            </clave>
         </indice>
         <indice>
```

```
            <ubicacion>prod.idx1</ubicacion>
            <tipo>1</tipo>
            <clave>
               <largo>4</largo>
               <posicion>120</posicion>
               <tipo>i</tipo>
            </clave>
         </indice>
         <indice>
            <ubicacion>prod.idx2</ubicacion>
            <tipo>1</tipo>
            <clave>
               <largo>10</largo>
               <posicion>8</posicion>
               <tipo>c</tipo>
            </clave>
         </indice>
      </indices>
      <claves_extranjeras>
         <clave_extranjera>
            <posicion>120</posicion>
            <referencia>rubro.rubro_idx0</referencia>
         </clave_extranjera>
      </claves_extranjeras>
   </archivo>
   <archivo>
      <ubicacion>rubro.dat</ubicacion>
      <registro>
         <largo>108</largo>
      </registro>
      <indices>
         <indice>
            <ubicacion>rubro.idx0</ubicacion>
            <tipo>0</tipo>
            <clave>
               <largo>4</largo>
               <posicion>4</posicion>
               <tipo>i</tipo>
            </clave>
         </indice>
         <indice>
            <ubicacion>rubro.idx1</ubicacion>
            <tipo>1</tipo>
            <clave>
               <largo>10</largo>
               <posicion>8</posicion>
               <tipo>c</tipo>
            </clave>
         </indice>
      </indices>
   </archivo>
</archivos>
```

Como se puede observar, son un conjunto de “tags” (<tag>…</tag>) relacionados. El tag “archivos” contiene una colección de tag’s “archivo”. Cada tag “archivo” describe un archivo con sus correspondientes índices. Un tag “archivo” tiene un tag “índices” que contiene una colección de tag’s “índice”, cada uno de los cuales describe un índice determinado. Se asume que el primer índice que aparece en el archivo es el índice 0, luego el 1 y así sucesivamente.

Para representar en código idxg al contenido del archivo xml anterior, deberíamos hacer las siguientes instrucciones:

```
int pos_clave = offsetof(struct pr,codigo);
int pos_rubro = offsetof(struct pr,rubro);
int pos_descr = offsetof(struct pr,descr);
int pos_clave_rb = offsetof(struct rb,codigo);
int pos_descr_rb = offsetof(struct rb,descr);
idxa *ar = idxg_inicio("prod.dat",sizeof(struct pr));
idxa *arrb = idxg_inicio("rubro.dat",sizeof(struct rb));
int idxcodigo = idxg_indice2(ar,"prod.idx0",pos_clave,IDXG_PK,'i');
int idxrubro = idxg_indice2(ar,"prod.idx1",pos_rubro,IDXG_SK,'i');
int idxdescr = idxg_indice(ar,"prod.idx2",10,pos_descr,IDXG_SK,'c');
int idxcod = idxg_indice2(arrb,"rubro.idx0",pos_clave_rb,IDXG_PK,'i');
int idxdes =
      idxg_indice(arrb,"rubro.idx1",10,pos_descr_rb,IDXG_SK,'c');
idxg_indicefk(db,arpr,"rubro","rubro_idx0",pos_rubro);
...
idxg_fin(ar);
idxg_fin(arrb);
```

como puede observarse, parece mucho más simple describir lo que queremos hacer utilizando un archivo xml más que utilizando código C escrito en términos de la librería idxg. Ahora con las funciones de manejo de archivos xml, el código puede escribirse de la siguiente forma:

```
idxdb *db = idxg_abrir("prod.xml");
idxa *ar = idxg_archivo2(db,"prod.dat");
idxa *arrb = idxg_archivo2(db,"rubro.dat");
...
idxg_cerrar(db);
```

Creación automática de archivo xml

Es posible decirle a idxg que cree el archivo xml por nosotros, a partir de un archivo y una serie de índices en memoria:

```
idxdb *db = idxg_nueva_bd("prod.xml",2);
```

```
idxa *arpr = idxg_archivo(db,0);
idxa *arrb = idxg_archivo(db,1);
int pos_clave = offsetof(struct pr,codigo);
int pos_rubro = offsetof(struct pr,rubro);
int pos_descr = offsetof(struct pr,descr);
idxg_inicio2(arpr,"prod.dat",sizeof(struct pr));
idxg_indice2(arpr,"prod.idx0",pos_clave,IDXG_PK,'i');
idxg_indice2(arpr,"prod.idx1",pos_rubro,IDXG_SK,'i');
idxg_indice(arpr,"prod.idx2",10,pos_descr,IDXG_SK,'c');
int pos_clave_rb = offsetof(struct rb,codigo);
int pos_descr_rb = offsetof(struct rb,descr);
idxg_inicio2(arrb,"rubro.dat",sizeof(struct rb));
idxg_indice2(arrb,"rubro.idx0",pos_clave_rb,IDXG_PK,'i');
idxg_indice(arrb,"rubro.idx1",10,pos_descr_rb,IDXG_SK,'c');
idxg_indicefk(db,arpr,"rubro","rubro_idx0",pos_rubro);

      idxg_grabar_xml(db);

      printf("creo_xml(): archivo prod.xml creado!\n");

idxg_cerrar(db);
free(db);
```

la función idxg_grabar_xml() genera el código del archivo xml por nosotros.

Creación manual de archivo xml

Lo habitual es tener que crear un archivo xml a partir de una simple estructura C que pretendemos grabar en un archivo y sobre el cual queremos crear una serie de índices para facilitar el acceso a los datos. En estos casos debemos crear el archivo xml en forma manual, utilizando cualquier archivo de texto (puede tomar el ejemplo de prod.xml y copiarlo). Completar los tags del archivo es simple, la dificultad puede estar en el tag “posicion” ¿Cuál es la posición de un determinado campo de una estructura C? Ejemplo, supongamos que deseamos averiguar la posición del campo “rubro” dentro de la estructura “pr”, para ello debemos utilizar la macro offsetof que está definida dentro de stddef.h :

```
#include <stdio.h>
#include <stdlib.h>
#include <stddef.h>
#include <idxg.h>

// estructura de registro de un producto
struct pr {
      char estado;
      int codigo;
      char descr[100];
```

```
        double precio;
        int rubro;
};

int main(int argc, char **argv) {
        int pos_rubro = offsetof(struct pr,rubro);
        printf("el campo rubro esta en posicion %d\n",pos_rubro);
        printf("el largo del campo rubro es %d\n",sizeof(int));
        printf("el largo de registro es %d\n",sizeof(struct pr));
        return 0;
}
```

Para saber el largo del campo, podemos utilizar sizeof(<tipo de dato del campo>), con esta información podemos completar los tags “posicion” y “largo”. El tag “tipo” referido al tipo de índice vale 0 (cero, que implica IDXG_PK, es decir, índice de clave primaria) o vale 1 (uno, que implica IDXG_SK, es decir, índice de clave secundaria). El tag “tipo” referido al tipo de clave del índice, puede adquirir los valores de tipo carácter (no es necesario poner entrecomillado) indicados en la tabla “Limites de esta librería” cuando se refiere a los tipos de índices implementados.

Compilación y Catalogación de esta Librería

Con el término catalogación nos referimos a la instalación de una librería o aplicación en un lugar determinado que permita su uso y normal funcionamiento. La librería idxg es una librería de tipo estática, con código binario que será incorporado a nuestro programa ejecutable.

Una librería o biblioteca debe compilarse –al igual que una aplicación- para obtener un código objeto compilado, ese código –junto con otros objetos compilados- debe guardarse en un archivo de código nativo de la biblioteca –el cual no necesariamente lleva el mismo nombre que el programa fuente- y por último, debe catalogarse en el directorio /lib para que este disponible para todas las aplicaciones. Para alcanzar este objetivo se crearon 2 proyectos geany, uno para Linux (idxg.geany) y otro para Windows (idxg_win.geany probado en Windows 7 utilizando MingW). El código C de esta librería está formado por 2 archivos: idxg.h y idxg.c; para compilar el código de la librería se deberá ejecutar el siguiente comando (ya sea en Linux o Windows):

```
gcc –Wall –c idxg.c –I<carpeta donde se encuentra idxg.h>
```

el parámetro –I (i latina mayúsculas, directorio de include’s) indica la carpeta en dónde el compilador C tiene que buscar los archivos .h (además de la búsqueda

que hace de los archivos .h de la biblioteca standard en los directorios de instalación del compilador). Como el programador tipeó **#include <idxg.h>** dentro de idxg.c entonces es necesario indicar la carpeta de include's.

Este comando va a generar el archivo idxg.o ; este archivo es utilizado por el comando **ar** (archive) para crear la librería con el siguiente comando:

```
ar rcs libidxg.a idxg.o
```

Este comando va a generar el archivo **libidxg.a** ; este archivo es la librería estática idxg.

Listo. Para distribuir esta librería, Ud. Necesita copiar en una carpeta 3 archivos:

- La librería idxg, es decir, el archivo libidxg.a
- Los prototipos de las funciones de la librería idxg, es decir, idxg.h
- El manual de la librería idxg

El procedimiento en Linux y Windows es el mismo, si Ud. utiliza MinGW. Junto con los archivos indicados, también se distribuye el archivo de comandos Windows buildwin.cmd , edite su contenido y allí podrá observar los comandos para compilar, construir y catalogar la librería en el directorio /lib , puede ejecutar el comando desde la ventana de comandos Windows (cmd.exe) estando posicionado en la carpeta donde se encuentra este archivo con solo escribir buildwin y presionar Enter.

Compilación y Linkedición de Aplicaciones que usan esta Librería

Para usar una librería debemos indicar –al menos- una sentencia #include indicando el nombre del archivo de cabecera (header) de la librería:

```
#include <idxg.h>
```

Esto se aplica a todas las librerías, incluso a las standards del lenguaje C, por ejemplo:

```
#include <stdlib.h>
```

Debe indicarle a GCC usando los switches[2] –I<directorio includes> (i mayúscula) la carpeta en donde se encuentra el archivo idxg.h. Algo similar sucede con el código nativo de las librerías, en la linkedición se incluyen automáticamente el código nativo de las librerías standards, pero debemos indicar el o los directorios en donde buscar el código nativo de las librerías propias o de terceros; para ello se utiliza el switch –L<directorio librerías>. Además de ello, debemos también indicar el nombre de la librería (puesto que en los directorios de librerías podría haber muchísimos archivos de librerías y examinarlos a todos haría muy lento el proceso de linkedición) utilizando el o los switch –l<librería> ("ele" minúscula). Por lo tanto, supongamos que el programa que va a ser uso de la librería idxg se llama testidxg.c y la carpeta en donde se encuentra la librería idxg (libidxg.a) y su archivo de cabecera (idxg.h) es d:\idxg\lib el comando para compilar el programa es:

```
gcc –Wall –c testidxg.c –Id:\idxg\lib
```

el comando para linkear (generar el ejecutable testidxg.exe) es:

```
gcc –Wall -o testidxg.exe testidxg.c -Ld:\idxg\lib -Id:\idxg\lib -lidxg
```

en el caso de Linux, los comando son idénticos, sólo deberá cambiar la ubicación de la carpeta d:\idxg\lib por la ubicación que corresponda.

Uso de idxg en Linux AMD64 (distribuciones de 64 bits)

La librería idxg ha sido desarrollada en Linux 32 bits, si Ud quiere utilizar la librería binaria (libidxg.a) en una distribución Linux de 64 bits, para el caso de una distribución basada en Linux Debian Ud. Deberá ejecutar el siguiente comando (como usuario root):

```
$ apt-get install gcc-multilib
```

Luego de instalar este paquete de software, vamos a poder compilar y linkear aplicaciones de 32 bits en un linux de 64 bits, para ello debemos agregar el "switch" –m32 al compilador gcc. Por ejemplo, supongamos que estamos posicionados en una carpeta cualquiera en donde esta nuestro programa testidxg.c que utiliza idxg y en la subcarpeta /lib se encuentra la librería idxg (idxg.h, libidxg.a, libidxg.pdf), el comando para compilar sería:

```
$ gcc -Wall -m32 -c testidxg.c -I./lib
```

[2] Se habla de "switches" porque el el switch –I puede repetirse n veces. Idem para el switch -L

el switch –I (“i” latina mayúscula) significa “directorio de include’s” (carpeta en donde gcc va a buscar los archivos .h no standards) y el commando para linkear sería

```
$ gcc -Wall -m32 -o testidxg testidxg.c -I./lib -L./lib -lidxg
```

testidxg es el nombre del programa ejecutable (en Windows debería ser testidxg.exe); el swtich –L./lib (“ele” mayúscula) significa “directorio de librerías” (carpeta en donde gcc va a buscar los archivos .a no standards); el switch –lidxg (“ele” minúscula) significa “nombre de librería a utilizar” (en este caso, gcc linkeará / mezclará nuestro código con la librería lib**idxg**.a para agregar el código binario de la misma y formar así el ejecutable). Estos switche’s pueden repetirse muchas veces en caso de utilizar distintas carpetas y librerías.

Límites de esta librería

Descripción	**Valor**
Tamaño máximo de largo en bytes de una clave de indexación	255
Cantidad máxima de registros de datos o de registros índices	Número máximo representado por un long con signo
Tamaño máximo del nombre y camino (ubicación) de archivo de datos	255 caracteres
Tamaño máximo del nombre y camino (ubicación) de archivo de índice	255 caracteres
Tamaño máximo del nombre y camino (ubicación) de archivo xml	255 caracteres
Tipos de índices implementados	‘c’ carácter, comparaciones binarias ‘s’ string, igual que carácter, pero realiza comparaciones no binarias de caracteres,’a’ y ‘A’ son iguales (case insensitive) ‘i’ integer, int ‘d’ double ‘l’ long