Estructuras de Datos y de la Información

Práctica 4

Búsqueda e inserción en ficheros

OBJETIVO: El objetivo de la práctica es implementar un tipo de datos abstracto que permita buscar e insertar registros en ficheros con distintos formatos de representación.

FECHA: Esta práctica se entregará los días 14, 15 y 16 de diciembre.

ENUNCIADO: Partiendo de la clase `Libro` y de los objetos `IOBuffer` de la práctica anterior, desarrollar un programa en lenguaje C++ que presente el siguiente menú de opciones:

```
1.- Abrir/crear fichero.
2.- Mostrar todos los registros.
3.- Buscar un registro.
4.- Insertar un registro.
5.- Cerrar.
0.- Salir.
```

El comportamiento del programa al seleccionar una opción será:

- En la opción 1 se solicita el nombre del fichero de trabajo. Si el fichero existe se abre en modo lectura/escritura y se le asocia un `IOBuffer` según su formato; si el fichero no existe se solicita el formato y se crea. El fichero permanecerá abierto en modo Lectura/Escritura hasta que se seleccione la opción 5.
- En la opción 2 se presentan por pantalla todos los registros del fichero de trabajo.
- En la opción 3 se solicita un valor de ISBN y se busca en el fichero de trabajo el registro cuyo campo ISBN coincide con el especificado. Si se encuentra el registro se visualiza por pantalla; en otro caso se muestra un mensaje de aviso.
- En la opción 4 se solicitan todos los campos de un libro. Se busca en el fichero de trabajo un registro con el mismo ISBN. Si no se encuentra, se inserta el nuevo registro al final; si se encuentra en el fichero un registro cuyo ISBN coincida con el introducido por teclado, se muestra un mensaje de aviso.
- En la opción 5 se cierra el fichero de trabajo.
- La opción 0 finaliza la ejecución del programa.

NOTAS DE IMPLEMENTACION:

- El fichero de trabajo `biblio.dat` contiene 148 registros del tipo `Libro` en formato 1 (registros de longitud fija con campos de longitud fija):

```
unsigned FechaEdicion;            // sizeof(unsigned)
char ISBN[TAM_ISBN];              // TAM_ISBN     =  11
char titulo[TAM_TITULO];          // TAM_TITULO   = 105
char autor[TAM_AUTOR];            // TAM_AUTOR    =  40
char libreria[TAM_LIBRERIA];      // TAM_LIBRERÍA =  20
```

- La familia de clases `IOBuffer` desarrollada en la práctica anterior permite trabajar con los siguientes formatos de fichero:
  1. Registros de longitud fija con campos de longitud fija
  2. Registros de longitud variable con indicador de longitud (`TSize`) y con campos separador por delimitador (`'|'`)
  3. Registros de longitud fija y campos de longitud variable con indicador de longitud (`char`)

- Utilizar un registro cabecera de 32 bytes para guardar un campo formato:
  `int Formato;` // Código del formato del fichero. El resto del registro cabecera no se utiliza.

Estructuras de Datos y de la Información

- Implementar el tipo abstracto `FileBuf` que permite trabajar con ficheros de registros organizados en campos según los formatos anteriores. Para ello, la clase `FileBuf` encapsula un objeto de tipo `IOBuffer` que maneja el formato, y encapsula un objeto `fstream` que maneja el acceso a ficheros de bytes.

```
class FileBuf {

  protected:              //Atributos
    fstream fs;           //Permite acceder a fichero de bytes
    IOBuffer *buf;        //Maneja el formato del fichero

     // Métodos protegidos para manejar el registro cabecera ...

  public:                //Métodos
  // Constructor
  FileBuf(){buf = 0; }

  // Destructor
  ~FileBuf() {if (buf) delete buf;}

  // Abre un fichero ya existente. En caso de error retorna un 0.
  // Lee el campo Formato del registro cabecera e instancia el objeto
  // IOBuffer adecuado.
  int FileBuf::Open(char *file, int mode = ios::in | ios::out |
                    ios::binary | ios::nocreate);

  // Crea un nuevo fichero con el formato especificado. Si ya existía lo
  // trunca. Retorna un 0 en caso de error. Instancia el objeto IOBuffer
  // del formato especificado.
  int FileBuf::Create(char *file, int format , int mode = ios::in |
                      ios::out | ios::binary | ios::trunc);

  // Si pos == PNULL, lee del fichero el siguiente registro;
  // Si pos >= 0, va a la dirección pos y lee un registro.
  // Retorna el número de bytes leidos del fichero.
  TSize FileBuf::Read(RECORD& r, TPointer pos = PNULL);

  // Si pos == PNULL, escribe un registro en el fichero;
  // Si pos >= 0, mueve el apuntardo a la posición indicada y escribe un
  // registro. Retorna el número de bytes escritos en el fichero.
  TSize FileBuf::Write(const RECORD& r, TPointer pos = PNULL);

  // Añade un nuevo registro al final del fichero.
  // Retorna el número de bytes escritos en el fichero.
  TSize FileBuf::Append(const RECORD& r);

  // Devuelve verdadero si el apuntador está al final del fichero.
  bool FileBuf::Eof(void) const;

  // Cierra el fichero de trabajo. Retorna 0 en caso de error.
  int FileBuf::Close(void);

  // Posiciona el apuntador al comienzo del fichero.
  void FileBuf::Rewind(void);
};
```

- Mediante la declaración de una plantilla, la clase `FileBuf` puede generalizarse para trabajar con cualquier tipo de registro (`RECORD`) y no sólo con registros tipo `Libro`.

```
template <class RECORD> class FileBuf {...};
```