Estructuras de Datos y de la Información

Práctica 3

Organización de Ficheros: Registros y Campos

OBJETIVO: El objetivo de la práctica es trabajar con diferentes formatos de representación de registros y campos.

FECHA: Esta práctica se entregará los días 30 de noviembre, 1 y 2 de diciembre.

ENUNCIADO: Realizar un programa C++ que realice copias de un fichero de registros en diferentes formatos. El programa solicita el nombre del fichero fuente, el nombre del fichero destino y el formato que se quiere dar al fichero destino.

Como fichero de trabajo se adjunta el fichero `biblio.dat` con 148 registros en formato de campos de longitud fija:

```
unsigned FechaEdicion;
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
```

Para el manejo del formato de los ficheros se implementará la familia de clases `IOBuffer`, al menos para los siguientes formatos:

1. Registros de longitud fija con campos de longitud fija
2. Registros de longitud variable con indicador de longitud y con campos separador por delimitador
3. Registros de longitud fija y campos de longitud variable con indicador de longitud

NOTAS DE IMPLEMENTACION:

- Implementar la clase `Libro` para encapsular los campos de información del registro y las operaciones que los manejan. Guardar la clase en los ficheros `libro.h` y `libro.C`.

- Sobrecargar los operadores de inserción y extracción de flujos/ficheros para la clase `Libro`.

```
istream& operator>> (istream& sin, Libro &r);
ostream& operator<< (ostream& sout, const Libro &r);
```

- Implementar la familia de clases `IOBuffer` para manejar diferentes formatos de registros en los ficheros `IOBuffer.h` y `IOBuffer.C`.

```
//
//  Fichero: iobuffer.h
//
#ifndef _IOBUFFER_H_
#define _IOBUFFER_H_
#include <fstream.h>

typedef long TPointer;
typedef int TSize;
const int MAXBYTES = 1000;
const long PNULL = -1;
```

```
// Clase Base IOBuffer
class IOBuffer {
  public:
    IOBuffer(): MaxBytes(MAXBYTES), BufferSize(0), NextByte(0){}
    virtual TPointer Read(fstream& fs) = 0;
    virtual TPointer Write(fstream& fs) const = 0;
    virtual TSize Pack(const void* field, TSize size = -1) = 0;
    virtual TSize UnPack(void* field, TSize size = -1) = 0;
    virtual void Clear(void) {BufferSize = NextByte = 0;}
    virtual TSize Length(void) const {return(BufferSize);}
  protected:
    char Buffer[MAXBYTES];
    TSize BufferSize;
    TSize MaxBytes;
    TSize NextByte;
};

// Buffer para manejar registros de longitud variable
class VarLenBuffer: public IOBuffer {
  public:
    TPointer Read(fstream& fs);
    TPointer Write(fstream& fs) const;
};

// Buffer para manejar registros de longitud fija
class FixLenBuffer: public IOBuffer {
  public:
    FixLenBuffer(TSize size): IOBuffer(), RecordSize(size) {}
    TPointer Read(fstream& fs);
    TPointer Write(fstream& fs) const;
  protected:
    TSize RecordSize;
};

// Buffer para manejar campos separados por delimitador
class DelimFieldBuffer: public VarLenBuffer {
  public:
    DelimFieldBuffer(char delim = '|'): VarLenBuffer(), Delim(delim) {}
    TSize Pack(const void* field, TSize size = -1);
    TSize UnPack(void* field, TSize size = -1);
  protected:
    char Delim;
};

// Buffer para manejar campos de longitud fija
class FixedFieldBuffer: public FixLenBuffer {
  public:
    FixedFieldBuffer(TSize size):  FixLenBuffer(size) {}
    TSize Pack(const void* field, TSize size);
    TSize UnPack(void* field, TSize size);
};

#endif
```

```
//  Fichero: iobuffer.C
//
#include "iobuffer.h"
#include <string.h>

//
// Mueve un registro de longitud variable del fichero al buffer.
// Retorna la posicion del registro en el fichero.
//
TPointer VarLenBuffer::Read(fstream& fs) {
  if(!fs.good()) return PNULL;
  TPointer pos = fs.tellg();

  Clear();
  fs.read(&BufferSize, sizeof(BufferSize));
  if(fs.fail() || (BufferSize > MaxBytes)) return PNULL;

  fs.read(Buffer, BufferSize);
  if(!fs.good()) {
    fs.clear();
    return PNULL;
  }
  return(pos);
}

//
// Mueve un registro de longitud variable del buffer al fichero.
// Retorna la posicion del registro en el fichero.
//
TPointer VarLenBuffer::Write(fstream& fs) const {
  TPointer pos = fs.tellp();
  fs.write(&BufferSize, sizeof(BufferSize));
  if(!fs.good()) return PNULL;
  fs.write(Buffer, BufferSize);
  if(!fs.good()) return PNULL;
  return(pos);
}

//
// Mueve un registro de longitud fija del fichero al buffer.
// Retorna la posicion del registro en el fichero.
//
TPointer FixLenBuffer::Read(fstream& fs) {
  if(!fs.good()) return PNULL;
  TPointer pos = fs.tellg();

  Clear();
  fs.read(Buffer, RecordSize);
  if(!fs.good()) {
    fs.clear();
    return PNULL;
  }
  return(pos);
}
```

```
//
// Mueve un registro de longitud fija del buffer al fichero.
// Retorna la posicion del registro en el fichero.
//
TPointer FixLenBuffer::Write(fstream& fs) const {
  TPointer pos = fs.tellp();
  fs.write(Buffer, RecordSize);
  if(!fs.good()) return PNULL;
  return(pos);
}

//
// Empaqueta un campo de longitud variable terminado en delimitador.
// Retorna el tamaño del campo empaquetado
//
TSize DelimFieldBuffer::Pack(const void *field, TSize size) {
  TSize len = strlen((char*)field);
  TSize start = NextByte;
  NextByte += len;
  if(NextByte >= MaxBytes) return -1;
  memcpy(&Buffer[start], field, len);
  Buffer[NextByte++] = Delim;
  if(BufferSize < NextByte) BufferSize = NextByte;
  return(len);
}

//
// Desempaqueta un campo de longitud variable terminado en delimitador.
// Retorna el tamaño del campo desempaquetado
//
TSize DelimFieldBuffer::UnPack(void *field, TSize size) {
  TSize start = NextByte;
  while((NextByte < BufferSize) && (Buffer[NextByte] != Delim)) NextByte++;
  if(NextByte > MaxBytes) return -1;
  TSize len = NextByte - start;
  memcpy(field, &Buffer[start], len);
  NextByte++;
  return(len);
}

//
// Empaqueta un campo de longitud fija.
// Retorna el tamaño del campo empaquetado
//
TSize FixedFieldBuffer::Pack(const void *field, TSize size) {
  TSize start = NextByte;
  NextByte += size;
  if(NextByte > MaxBytes) return -1;
  memcpy(&Buffer[start], field, size);
  if(BufferSize < NextByte) BufferSize = NextByte;
  return(size);
}

//
// Desempaqueta un campo de longitud fija.
// Retorna el tamaño del campo desempaquetado
//
TSize FixedFieldBuffer::UnPack(void *field, TSize size) {
  TSize start = NextByte;
  NextByte += size;
  if(NextByte > MaxBytes) return -1;
  memcpy(field, &Buffer[start], size);
  return(size);
}
```