Prácticas Memoria Secundaria

Enunciado 1: Organización Secuencial

La Organización Secuencial es la forma básica de almacenar un conjunto de registros que constituyen un archivo. En ella, los registros quedan grabados consecutivamente cuando el archivo se crea y debe accederse a ellos de forma consecutiva cuando el archivo se procesa.

En esta práctica se estudia el acceso secuencial a ficheros y se implementarán algunos de las funciones básicas que permiten gestionar este tipo de ficheros.

TAREAS A REALIZAR POR EL ALUMNO

Partiendo del archivo secuencial (alumnos.dat) que contiene varios registros de tipoAlumno tal como se define en secuencial.h, implementar las siguientes funciones que permiten el acceso y la gestión del fichero secuencial:

1. Implementar la función

int leeSecuencial(char *fichSecuencial)

que toma como entrada el nombre del fichero secuencial que se desea leer (alumnos.dat) y muestra el contenido del mismo. La función devuelve el número de registros que tiene el fichero y muestra todos los registros del mismo anteponiendo a cada uno de ellos el número relativo del registro (NRR) en el fichero.

2. Implementar la función

int buscaReg(FILE *f, tipoAlumno *reg, char *dni)

que busque secuencialmente en el archivo un registro a partir de su clave, parámetro dni de entrada a la función. Si el registro se encuentra en el archivo la función debe devolver el NRR en el fichero y el registro completo en el parámetro *reg*. Si el registro no está en el archivo la función debe devolver el valor -1 indicando que no existe un registro con esa clave.

3. Implementar la función

int insertaReg(char *fichEntrada, tipoAlumno *reg)

que inserte al final del fichero secuencial el registro cuya información contiene el parámetro *reg*. Si la inserción se produce con éxito la función debe devolver un entero que indique el NRR del nuevo registro insertado y -1 si se produce un error.

AYUDA PARA LA REALIZACIÓN DE LA PRÁCTICA

El fichero pruebaSecuencial.c muestra un ejemplo de utilización de algunas de estas funciones. En el fichero resultadoPruebaSecuencial.txt se muestra la salida de la ejecución del ejemplo.

Enunciado 2: Organización de archivos por el Método de Dispersión

Partiendo del archivo (**alumnos.dat**) que contiene varios registros de tipoAlumno tal como se define en **dispersion.h**, se debe construir un nuevo archivo (***alumnos.hash***) que organice estos registros según el Método de Dispersión.

Para conseguir esta organización, se deben seguir los siguientes criterios:

1. El fichero estará formado por **20 CUBOS** con **CAPACIDAD** para **5** registros de tipoAlumno cada uno.
2. Puesto que algunos de estos 20 cubos iniciales pueden llenarse se añadirán **4 CUBOS DE DESBORDE** más para almacenar los registros que se asignen a cubos LLENOS.
3. Cada cubo debe almacenar además el número de registros que se le han asignado, incluyendo los que no haya podido almacenar porque estaba lleno.
4. Los tres criterios anteriores se reflejan en la definición del tipoCubo que se proporciona en archivos.h y que es obligatorio utilizar.
5. Los registros desbordados se irán almacenando **secuencialmente** en el área de DESBORDE. Los 5 primeros registros desbordados irán al primer cubo de desborde, los 5 siguientes al siguiente cubo de desborde, etc.
6. Utilizar como **clave** el campo DNI de los registros de tipoAlumno y aplicar la función **Módulo** como función hash para asignar un registro a un cubo. Previamente habrá que convertir el DNI a entero, por tanto la función que se aplica para obtener el cubo asignado a un registro será

f(k)=atoi(K)%CUBOS

TAREAS A REALIZAR POR EL ALUMNO

Siguiendo los criterios descritos al inicio de la práctica

1. Implementar la función

int creaHash(char *ficheroEntrada,char *fichSalida)

que toma como entrada el nombre del fichero que se desea organizar (alumnos.dat) y el nombre del fichero que va a crear (alumnos.hash) utilizando el método de dispersión. La función debe devolver un entero que indique el número total de registros desbordados.

2. Implementar la función

int buscaReg(FILE *f, tipoAlumno *reg, char *dni)

que busque en el archivo creado un registro a partir de su clave, parámetro **dni** de entrada a la función. Si el registro se encuentra en el archivo la función debe devolver el número de CUBO en que está almacenado y el registro completo en el parámetro **reg**. Si el registro no está en el archivo la función debe devolver el valor -1 indicando

que no existe un registro con esa clave.

La función de búsqueda debe seguir los mismos criterios, por tanto, inicialmente debe buscar el registro en el cubo que le asigne la función hash; si no esta allí y el cubo está desbordado habrá que buscarlo secuencialmente en el área de desborde. Si no está en ese cubo y el cubo no se ha llenado todavía no hay que seguir buscando, el registro no está en el archivo.

3. Implementar la función

int insertarReg(FILE *f, tipoAlumno *reg)

que inserte en el fichero el registro que se le pasa como parámetro. La función debe devolver el número real del cubo en el que se inserta el registro y -1 si el cubo asignado inicialmente se ha desbordado y los 4 CUBOS DE DESBORDE ya se han llenado, indicando que el registro no se ha podido almacenar y hay que reorganizar el archivo.

AYUDA PARA LA REALIZACIÓN DE LA PRÁCTICA

En el fichero dispersion.c se proporciona:

1. Una función que genera un archivo vacío organizado siguiendo los criterios descritos anteriormente.
2. Una función que lee cualquier archivo organizado según los criterios anteriores mostrando el contenido de sus cubos y el número de registros que se le han asignado a cada uno.

El fichero pruebaDispersion.c muestra un ejemplo de utilización de estas funciones y de otras que el alumno debe codificar como se indica en las tareas previas. En el fichero resultadosPruebaDispersion.pdf se muestra la salida de la ejecución del ejemplo y en él puede observarse como quedan organizados los registros del fichero de entrada en el fichero de salida.

Normas generales

En la realización de las prácticas se deben seguir los siguientes criterios:

1. Es **obligatorio** utilizar los tipos y prototipos siempre que se proporcionan en el fichero cabecera. Si no se proporcionan debe crearse el fichero cabecera correspondiente o añadir al proporcionado los prototipos de todas las funciones que se implementen.
2. La codificación de las funciones se realizará en un fichero fuente diferente al del programa principal.
3. Para visualizar el correcto funcionamiento de las operaciones implementadas se debe crear un fichero de prueba desde el cual se llamará a las funciones implementadas.
4. Se debe crear el correspondiente Makefile para la correcta creación del fichero ejecutable.