Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

Ingeniero en Software y tecnologías emergentes

Materia: Programación Estructurada / Clave **36276**

Alumno: Antonio Ramos González

Matrícula: 372576

Maestro: Pedro Núñez Yépiz

Actividad No. : 14

Tema - Unidad : Archivos

Ensenada Baja California a 27 de Noviembre del 2023

1. INTRODUCCIÓN

En el ámbito de la programación, la eficiente manipulación de datos se vuelve esencial para mejorar el rendimiento. La habilidad para trabajar con archivos binarios y la implementación de índices son elementos fundamentales que permiten optimizar la gestión de información a gran escala. Estas prácticas no solo facilitan el acceso y la modificación de datos, sino que también contribuyen significativamente a la eficiencia y velocidad en el procesamiento de información en entornos computacionales.

2. COMPETENCIA

Facilitar al estudiante la capacidad de manipular archivos binarios y crear índices para mejorar la eficiencia en la carga de memoria, además de asegurar un manejo competente de estos recursos.

3. FUNDAMENTOS

Los archivos binarios son un tipo de archivo que su contenido este encriptado de forma que no se puede modificar de forma externa.

Para acceder a ellos se hace de la misma manera que los archivos de texto

```
File *ap;

Ap=fopen("Nombre.dll", "rb");
```

A la hora de querer leer o escribir dentro de un archive binario se utilizará

```
Treg temp;

fread(&temp, sizeof(Treg), 1, ap);

fwrite(&temp, sizeof(Treg), 1, ap);
```

4. PROCEDIMIENTO

ACTIVIDAD 14
Archivos Binarios
(archivos indexados)

MENÚ

1.- AGREGAR
2.- ELIMINAR
3.- BUSCAR
4.- ORDENAR
5.- IMPRIMIR REGISTROS ARCHIVO ORIGINAL
6.- IMPRIMIR REGISTROS ARCHIVO ORDENADO
7.- GENERAR ARCHIVO TEXTO
8.- EMPAQUETAR
0.- SALIR

INSTRUCCIONES: Programa que contenga el menú anterior, el programa utiliza un vector de índices de la siguiente estructura: **[llave, índice]** donde *el campo llave es* **noempleado. registros.dat** es el archivo con los registros a cargar en el vector de índices **archivo binario sera proporcionado**,

CARGAR ARCHIVO : El programa deberá cargar al arrancar el programa, el archivo Binario generará el vector de índices (**llave, indice**) sólo con registros válidos **(el tamaño del vector debera ser 25% mas grande que el la cantidad de registros que contenga el archivo binario) utiliza un archivo externo para averiguar tamaño y retorne cantidad de registros.**

1.- Agregar :
El programa deberá ser capaz de agregar un registro al arreglo de índices y al final del archivo Binario. (agregar forma automatica no repetido el campo llave)

2.- Eliminar :

El programa deberá buscar una **noempleado** en el vector de índices por medio del método de búsqueda más óptimo.
La función deberá *retornar*, el índice donde se encuentra la matrícula en el archivo Binario, utilizar banderas para escoger el método más adecuado.
Una vez obtenido el índice moverse dentro del archivo binario (usar fseek) usando el índice del vector de índices.
Leer el registro en la posición correcta, preguntar si se quiere eliminar registro.
Cambiar el status del registro si la respuesta es afirmativa, volver a posición anterior y sobreescribir el registro.

3.- Buscar :
El programa deberá buscar un **noempleado** en el vector de índices por medio del método de búsqueda más óptimo.
La función deberá *retornar*, el índice donde se encuentra la matrícula en el archivo Binario, utilizar banderas para escoger el método más adecuado.
Una vez obtenido el índice moverse dentro del archivo binario (usar fseek) usando el índice del vector de índices.
Leer el registro en la posición correcta, y desplegar el registro.

4.- Ordenar :
El programa deberá ordenar el vector de índices por medio del método de ordenación más óptimo. Utilizar banderas para escoger el método más adecuado por el que se ordenará por el campo llave (**noempleado**) o no ordenarse si ya está ordenado. (utilizar 3 metodos de ordenacion diferentes segun sea el caso que se necesite Justificar los metodos en el reporte)

5 Y 6.- Mostrar Todo:
El programa deberá mostrar todos los registros del Archivo Binario, preguntar: **ordenado o normal**. Usar el vector de índices para imprimirlo ordenado, y directamente desde el archivo si es normal.

7.- GENERAR ARCHIVO TEXTO:
El programa deberá generar un archivo de texto, el usuario debe proporcionar el nombre del archivo.
El programa deberá mostrar todos los registros del Archivo Binario, preguntar: **ordenado o normal**. Usar el vector de índices para imprimirlo ordenado, y directamente desde el archivo si es normal.
el programa podrá generar múltiples archivos para comprobar las salidas.

8.- EMPAQUETAR :
El programa deberá actualizar el Archivo Binario, a partir de solo registros válidos, y eliminarlos del archivo binario. Crear copia y archivo de respaldo .bak del archivo de antes de eliminarlos.

5. RESULTADOS Y CONCLUSIONES

Los archivos binarios un tipo de archivo indispensable para el almacenamiento de datos importantes, debido a que a pesar de que se pueda acceder a ese archivo, no se puede cambiar u contenido de ninguna manera.

```c
/*
Antonio Ramos Gonzalez Mt: 372576
24/11/2023 || 27/11/2023
El ususario podra agregar, eliminar, buscar, ordenar, imprimir el registro original y el ordc
RGA_Act14_932
*/
#include "Babilonia.h"
typedef int Tkay;

int msg();
void menu(int numReg);
int cargar();
void cargar2(TWrKr reg[], Tindex index[]);
TWrKr agregar(TWrKr reg[], int n);
int searchSec(Tindex index[], int n, int mt);
void printReg(TWrKr reg);
void eliminar(int n);
void buscar(int n);
void agregarbin(TWrKr reg);
int orderBu(Tindex reg[], int n);
void printOrder(Tindex indice[], int n);
void printOriginal(int n);
void crearTxtOrig();
void crearTxtOrd(Tindex indice[], int n);
void Empaqueta(int n);
void quicksort(Tindex reg[], int low, int high);
void swap(Tindex reg[], int i, int j);
int partition(Tindex reg[], int low, int high);
void insertionSort(Tindex reg[], int n);
int searchBin(Tindex reg[], int inf, int sup, int mt);
void printRegOr(Tindex reg);

> int main() ···

> int msg() ···

> void menu(int numReg) ···

> int cargar() ···

> void cargar2(TWrKr reg[], Tindex index[]) ···

> TWrKr agregar(TWrKr reg[], int n) ···

> int searchSec(Tindex index[], int n, int mt) ···

> void printReg(TWrKr reg) ···

> void printRegOr(Tindex reg) ···

> void eliminar(int n) ···

> void agregarbin(TWrKr reg) ···

> int orderBu(Tindex reg[], int n) ···

> void printOrder(Tindex indice[], int n) ···

> void printOriginal(int n) ···

> void crearTxtOrig() ···

> void crearTxtOrd(Tindex indice[], int n) ···

> void Empaqueta(int n) ···

> void quicksort(Tindex reg[], int low, int high) ···

> int partition(Tindex reg[], int low, int high) ···

> void swap(Tindex reg[], int i, int j) ···

> void insertionSort(Tindex reg[], int n) ···

> int searchBin(Tindex reg[], int inf, int sup, int mt) ···
```

6. ANEXOS
RGA_RP11 _PE(ANEXO)

7. REFERENCIAS

Diseño de algoritmos y su codificación en lenguaje C

Corona, M.A. y Ancona, M.A. (2011)..

España: McGraw-Hill.

ISBN: 9786071505712

Programación estructurada a fondo:implementación de algoritmos en C

:Pearson Educación.Sznajdleder, P. A. (2017)..

Buenos Aires,Argentina: Alfaomega

Como programar en C/C++

H.M. Deitel/ P.J. Deitel

Segunda edición

Editorial: Prentice Hall.

ISBN:9688804711

Programación en C.Metodología, estructura de datos y objetos

Joyanes, L. y Zahonero, I. (2001)..

España:McGraw-Hill.

ISBN: 8448130138