Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

Ingeniero en Software y tecnologías emergentes

Materia: Programación Estructurada / Clave **36276**

Alumno: Fernando Haro Calvo

Matrícula: 372106

Maestro: Pedro Núñez Yépiz

Actividad No. 13

Tema - Unidad 1: Archivos Binarios

Ensenada Baja California a 12 de noviembre del 2022

Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

1. INTRODUCCIÓN

En esta práctica, se utilizarán archivos binarios de una manera mucha más eficiente que en prácticas pasadas, optimizando nuestro uso de memoria principal usando un arreglo índice para acceder a la información más grande contenida en el archivo. Consiste en generar aleatoriamente información de empleados para llenar un archivo de datos de personas, donde también podremos realizar métodos de búsqueda y ordenación, al igual que imprimir los registros en la consola en forma de tabla o de registro y poder generar un archivo de texto con estos datos. Además, el programa deberá mantener registro de las personas en el vector, ya sean activos o inactivos.

2. COMPETENCIA

El objetivo principal poner en práctica los conocimientos adquiridos en programación en C, especialmente abordará:

Archivos Binarios: El programa deberá poder leer y escribir archivos binarios, al igual que renombrarlos usando extensiones ".dat" y ".bak" para almacenar información de un vector usando una estructura específica.

Archivos de Texto: El programa deberá aplicar las estrategias de escribir, anidar y leer archivos de texto en formato ".txt" para almacenar información.

Estructuras de dato 'struct': Se utiliza para almacenar diferentes tipos de datos en una sola entidad.

Algoritmos de búsqueda (secuencial y binaria): se refiere a la capacidad de diseñar e implementar algoritmos que permitan encontrar un elemento específico en una colección de datos, como un arreglo.

Validación de Datos: El programa debe ser capaz de validar los datos de entrada, como el nombre, fecha de nacimiento, sexo y entidad federativa, para asegurarse de que sean correctos y cumplan con los requisitos oficiales.

Modularidad y Organización: La práctica debe demostrar una estructura modular y organizada del código, utilizando funciones y procedimientos para dividir el problema en tareas más pequeñas y manejables.

Excepciones y Control de Errores: El programa debe ser robusto, manejando excepciones y errores de manera adecuada. Debe ser capaz de informar al usuario si se ingresan datos incorrectos y nunca terminar inesperadamente.

3. FUNDAMENTOS

Programación en C: La práctica se basa en la programación en el lenguaje C, que es ampliamente utilizado en el desarrollo de sistemas y aplicaciones de software.

Funciones: Las funciones son bloques de código reutilizables que realizan tareas específicas. La práctica se enfoca en la creación y uso de funciones para organizar y modularizar el código.

Structs: Son una colección de variables llamadas "miembros" que pueden tener diferentes tipos de datos, como enteros, caracteres, flotantes u otros structs. Los structs se utilizan para representar una entidad o conjunto de datos que tiene múltiples atributos relacionados.

Cadenas: Son una secuencia de caracteres, es decir, una serie de caracteres que se organizan de manera consecutiva. Estos caracteres pueden ser letras, números, símbolos y espacios en blanco.

Validación: Es una parte esencial de la programación para garantizar que los datos sean correctos y seguros, y para evitar errores o comportamientos inesperados en el programa.

Generación de archivos ".txt": Utilizando los datos de un struct, generar una archivo de texto usando las funciones de la librería estándar de C.

Generación de archivos binarios: Utilizando los datos de un struct, generar un archivo binario usando las funciones de la librería estándar de C para mantener encriptados los datos.

4. PROCEDIMIENTO

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 será proporcionado,

CARGAR ARCHIVO: El programa deberá cargar al arrancar el programa, el archivo Binario generará el vector de índices (llave, índice) sólo con registros válidos (el tamaño del vector deberá ser 25% más grande que 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 automática 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 sobrescribir 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 métodos de ordenación diferentes según sea el caso que se necesite Justificar los métodos 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

La actividad fue de gran ayuda, ya que se logró optimizar en gran medida el uso de memoria comparado con programas y actividades anteriores realizadas, donde teníamos en todo momento cargado un vector de datos gigante, tomando una gran cantidad de memoria y poniendo en mayor riesgo la integridad de los datos.

En cuanto a métodos de ordenamiento, se agregó uno más en el caso de que los datos sean mayores a 1000, el merge sort, un algoritmo de ordenamiento recursivo más rápido para grandes cantidades de datos que el quick sort, además se cambió la condición para usar el bubble sort mejorado, siendo su uso cuando el arreglo de índices está “casi ordenado”, es decir cuando existen pocos elementos fuera de orden, resultando en un proceso mucho más eficiente.

```
int ordOpt(Tindice vect[], int n, int ordenado)
{
    int i;

    if (ordenado == 0)
    {
        if (n <= 1000)
        {
            i = ordQuick(vect, n);
            printf("Ordenado usando Quick Sort\n");
        }
        else
        {
            i = ordMerge(vect, n);
            printf("Ordenado usando Merge Sort\n");
        }
    }
    else
    {
        if (ordenado == 2)
        {
            i = ordBubbleMejorado(vect, n);
            printf("Ordenado usando Bubble Sort Mejorado\n");
        }
    }

    return 1;
}
```

En resumen, la práctica me ayudó a lograr un manejo más eficiente de archivos de texto, archivos binarios, vectores, índices, métodos de búsqueda y especialmente de ordenación, cumpliendo con las recomendaciones establecidas.

6. ANEXOS

Código y salidas: HCF_ACT14_ANEXOS
Archivos: HCF_ACT14_932.c
Librería: alexandria.h

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