Universidad Autónoma de Baja California

Facultad de Ingeniería, Arquitectura y Diseño F.I.A.D.

Campus Ensenada

Ramón Mejía Martínez 372099

Grupo: 932

Actividad 14

Programación Estructurada

Archivos Binarios (Indexados)

Ensenada, Baja California a 28 de noviembre de 2023.

Universidad Autónoma de Baja California

Facultad de Ingeniería Arquitectura y Diseño

Ingeniero en Software y tecnologías emergentes

Materia: Programación Estructurada / Clave **36276**

Alumno: Ramón Mejía Martínez

Matrícula: 372099

Maestro: Pedro Núñez Yépiz

Actividad No. : 14

Tema - Unidad : Archivos Binarios Indexados

Ensenada Baja California a 28 de noviembre del 2023.

1. Introducción

En esta actividad se realizará un sistema de gestión de empleados usando structs, archivos de texto y binarios, usando como almacenamiento principal un archivo binario indexado para realizar distintas operaciones en el archivo.

2. Competencia

Esta actividad tiene como objetivo el manejo cadenas, librerías y funciones, structs, métodos de ordenación y búsqueda, uso de archivos indexados como almacenamiento principal, con el fin de contar con mas herramientas que nos pueden ser útiles para optimizar almacenamiento a cambio de rendimiento, para poder tomar la decisión según lo que se este programando y cual sea el tipo de máquinas que correrán el programa.

3. Fundamentos

Archivos

Archivo. Es un conjunto de información almacenada bajo un nombre. Cada archivo es almacenado en un dispositivo de almacenamiento, usando un único nombre. El dispositivo puede ser: el disco floppy, el disco duro, cd-rom, etc. La librería contiene las funciones para procesar archivos.

Comandos para manjar archivos.

La función remove elimina el archivo con el nombre que se le mande como argumento.

Remove(“nombre”);

La función rename cambia el nombre de un archivo ya existente.

Rename(“nombre_antiguo”,”nombre_nuevo”);

Estos son los que son usados para esta actividad.

Archivos de texto

Operaciones:

Los modos de operación son: r, w y a, a los cuales se les puede agregar un +.

- El modo de operación a se usa para leer y escribir, devuelve la dirección de memoria del archivo si existe, si no existe se crea. Coloca el cursor en la siguiente posición del último carácter del archivo para continuar escribiendo en el archivo.

- w se usa para escribir, devuelve la dirección de memoria del archivo si existe, si no existe se crea, se usa w+ para poder leer y escribir. Coloca el cursor al inicio del texto y reemplaza los datos del archivo si ya existía.

- r se usa para lectura, devuelve la dirección de memoria del archivo si se encontró o un valor nulo si no existe, se usa r+ para poder leer y escribir del archivo. El puntero al abrir un archivo de esta forma se encuentra en el inicio del archivo.

- Abrir.

Para abrir un archivo binario se define un puntero con tipo de dato FILE y se usa la instrucción fopen.

FILE *fa;

fa=fopen(“nombre.extension”,”modo_de_operacion”);

- Leer.

Para leer datos de un archivo de texto se usa fscanf.

scanf(puntero_a_archivo,”datos extraidos”,variable);

Leyendo y guardando en variables, si solo se desea mostrar el archivo se puede guardar todo un una sola variable usando fgets y mostrar carácter por carácter hasta llegar al final del archivo.

- Escribir.

Para escribir en un archivo de texto se usa fprintf.

fprintf(puntero_a_archivo,”texto_o_tipoDeDato”,variable);

Archivos binarios

Operaciones:

Las operaciones para los archivos binarios son iguales a las de los archivos de texto y se abren de la misma manera con la excepción de que se tiene que indicar al lado del modo una b de binario (fa=fopen(“nombre.dll”,”rb”);

- Leer

Para leer de un archivo binario se usa fread.

fread(&variable,sizeof(tipo_dato_variable),numero_de_registros,puntero_a_archivo);

Los datos se guardan en la variable.

Requiere saber el tamaño de un registro del archivo binario para poder dividir correctamente la información.

Normalmente de leen de 1 en 1 los registros.

- Escribir

Para escribir en un archivo binario se usa fwrite.

fwrite((&variable,sizeof(tipo_dato_variable),numero_de_registros,puntero_a_archivo);

Usa los mismos argumentos, la diferencia es que ahora se escribe lo de la variable en el archivo.

Otras funciones útiles son feof(puntero_a_archivo) que indica cuando se llega al final del archivo.

Al terminar de trabajar con los archivos se deben cerrar para mantener la integridad de los datos, esto se hace usando fclose(puntero_a_archivo).

Arreglos indexados.

Este tipo de arreglos contienen los índices y otros campos esenciales para identificar un registro, de esta manera se usa un archivo como almacenamiento principal y se realizan operaciones como ordenamiento, búsqueda, etc. Al mostrar al usuario únicamente se hace la consulta al archivo a través del arreglo indexado.

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, índice**) sólo con registros válidos **(el tamaño del vector deberá ser 25% más 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 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 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.

NOTA: Programa 100% Validado.
NOTA2: usar libreria propia con las funciones más comunes

ARCHIVO datos.dat e Imagen de la estructura usada en datos.dat

5. Resultados y conclusiones:

Constantes y Structs usadas.

ELIMINAR Y BUSCAR se usa al imprimir un registro, se manda la razón para realizar la opción correspondiente.

Se agrego teléfono al arreglo indexado para poder verificar que no se repitan los números.

```
#define ELIMINAR 1
#define BUSCAR 0
#define ENCABEZADO "| NO   | NO EMP | PUESTO                | APELLIDO PATERNO |
typedef int Tkey;

typedef struct _Wrkr{
    int status;
    Tkey key;
    char name[L];
    char apPat[LN];
    char apMat[LN];
    char sexo[15];
    char puesto[L];
    char estado[L];
    int edad;
    Tkey telefono;
}TWrkr;


typedef struct _index{
    Tkey key;
    Tkey indice;
    Tkey Telefono;
}Tindex;
```

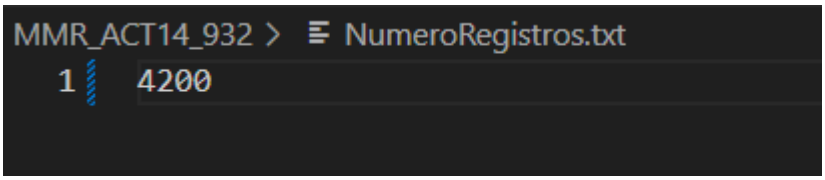
Programa para determinar tamaño del arreglo indexado

Se usa la librería <sys/stat.h> de la cual se usa la struct stat, que guarda información de un archivo, y la función stat, que recopila esa información.

Simplemente se divide el tamaño total por el tamaño del struct usada y se multiplica por 1.25. Esto se guarda en un archivo para poder recibirlo en el programa principal.

```c
#include <stdio.h>
#include <sys/stat.h>
```

```c
int main()
{
    FILE *fa;
    int cantidad;
    struct stat archivo;
    stat("datos.dat",&archivo);
    cantidad=archivo.st_size/sizeof(TWrkr);
    cantidad*=1.25;
    fa=fopen("NumeroRegistros.txt","w");
    fprintf(fa,"%d",cantidad);
    fclose(fa);
    return 0;
}
```



Función main:

En la función main se manda a ejecutar el programa para determinar el tamaño del arreglo de índices.

```
int main()
{
    srand(time(NULL));
    FILE *fa;
    int REG;
    system("C:\\Users\\ramon\\OneDrive\\Escritorio\\MMR_ACT1_932\\MMR_A
    fa=fopen("NumeroRegistros.txt","r");
    if(fa)
    {
        fscanf(fa,"%d",&REG);

        fclose(fa);
        menu(REG);
    }
    else
    {
        printf("\nEl archivo binario de registros no existe\n");
        system("pause");
    }

    return 0;
}
```

Función menú:

- Recibe el tamaño generado por el programa mencionado.

Al iniciar el menú se define el arreglo indexado y lo inicializa además de determinar el método de ordenación.

```
void menu(int REG)
{
    int i=0,j=0,respaldo=0;
    REG;
    int op;
    int orden=0,MetodoOrden=0;
    Tindex trabajadores[REG];
    TWrkr reg;
    FILE *fa;
    fa=fopen("datos.dat","rb");
    if(fa)
    {
        //carga de archivo binario al arreglo indexado
        while(fread(&reg,sizeof(TWrkr),1,fa)==1)
        {
            if(reg.status)
            {
                trabajadores[i].indice=j;
                trabajadores[i].key=reg.key;
                trabajadores[i].Telefono=reg.telefono;
                i++;
            }
            j++;
        }
        fclose(fa);
        if(i>3360)
        {
            if(i>4000)
            {
                MetodoOrden=2;
            }
```

Se lee la opción, se llama a las funciones correspondientes y se asignan las banderas según lo que se hizo.

Al agregar algún empleado se vuelve a verificar el método de ordenación adecuado.

```
switch(op)
{
    case 1:
        if(i+1<REG)
        {
            GenWrkr(trabajadores,i++);
            orden=0;
            if(i>3360)···
        }
        else···
        break;
    case 2:
        DelWrkr(trabajadores,i,orden);
        break;
    case 3:
        BuscarWrkr(trabajadores,i,orden);
        break;
    case 4:
        if(!orden)
        {
            OrdenarIndex(trabajadores,i,MetodoOrden);
            orden=1;
        }
        else
        {
            printf("\nEl arreglo indexado ya esta ordenado\n");
        }
        break;
    case 5:
        PrintRegTab(trabajadores,i,orden,MetodoOrden);
```

Agregar

Función GenWrkr:

Genera valores aleatorios a un registro y lo agrega al archivo binario.

- Recibe el arreglo indexado y el número de empleados activos.

```
void GenWrkr(Tindex vect[],int i)
{
    FILE *fa;
    TWrkr reg;
    int s,num;
    int j;
    reg.status=1;
    reg.key=GenKey(vect,i,300000,399999);
    vect[i].key=reg.key;
    vect[i].indice=i;
    reg.edad=(rand()%53)+18;
    GenEst(reg.estado);

    s=rand()%2;

    if(s)
    {
        strcopy(reg.sexo,"HOMBRE");
        GenH_Name(reg.name);
    }
    else…

    GenAp(reg.apPat,reg.apMat);

    do
    {
        num=(rand()%100000)+(rand()%99)*100000;
    }while(BuscarSecTelIndex(vect,i,num)!=-1);
    reg.telefono=num;
```

```
fa=fopen("datos.dat","rb");
if(fa)
{
    fclose(fa);
    fopen("datos.dat","ab");
    fwrite(&reg,sizeof(TWrkr),1,fa);
    fclose(fa);
    printf("\nSe genero un trabajador y se guardo en el archivo\n");
}
else…
```

Funciones de generación aleatoria.

Función GenH_Name:

Función GenM_Name:

Función GenAp:

Función GenPuesto:

Función GenEst:

Estas cinco funciones trabajan de la misma manera.

- Generan un nombre, apellido, puesto o estado de manera aleatoria, según corresponda.
- Reciben el campo del registro donde se guardan los datos.
- Copia el nombre del índice aleatoria de la lista y lo asigna al campo correspondiente.

```
void GenH_Name(char str[])
{
    char h_name[100][LN]={···
    int prob;
    prob=rand()%10;
    if(prob<=7)
    {
        strcopy(str,h_name[rand()%100]);
    }
    else
    {
        strcopy(str,h_name[rand()%100]);
        strConcat(str,h_name[rand()%100]);
    }
}
```

Función GenKey:

Genera un numero de empleado que no exista en el arreglo de índices.

- Recibe el arreglo de índices, numero de empleados activos, rango valido del número de empleado.
- Devuelve el número generado.

Se realiza el mismo proceso para generar el número de teléfono.

```
int GenKey(Tindex vect[],int i,int ri,int rf)
{
    int num;
    int rango=rf-ri;
    do
    {
        num=(rand()%rango+1)+ri;
    }while(BuscarSecIndex(vect,i,num)!=-1);
    return num;
}
```

Eliminar

Función DelWrkr:

Busca y elimina a un empleado activo si así se decide.

- Recibe el arreglo de índices, número de empleados activos y la bandera que indica si el arreglo de índices esta ordenado.

```
void DelWrkr(Tindex vect[],int i,int orden)
{
    int pos;
    int del;
    FILE *fa;
    printf("Eliminar empleado\n\n");
    pos=validNum("Ingresa el número de empleado que deseas eliminar: ",
    if(orden)
    {
        pos=BuscarBinIndex(vect,i,pos);
        if(pos!=-1)
        {
            PrintRegReg(pos,ELIMINAR);
        }
        else
        {
            printf("\nEl empleado no existe\n");
        }
    }
    else…
```

Buscar

Función BuscarWrkr:

Busca un empleado y muestra sus datos.

- Recibe el arreglo de índices, número de empleados activos y la bandera que indica si el arreglo de índices esta ordenado.

```
int BuscarWrkr(Tindex vect[],int i,int orden)
{
    int pos;
    FILE *fa;
    printf("Buscar empleado\n\n");
    pos=validNum("Ingresa el número de empleado que deseas buscar:
    if(orden)
    {
        pos=BuscarBinIndex(vect,i,pos);
        if(pos!=-1)
        {
            PrintRegReg(pos,BUSCAR);
        }
        else
        {
            printf("\nEl empleado no existe\n");
        }
    }
    else ···
```

Función PrintRegReg:

Muestra el registro de un índice consultando el archivo binario y si la razón es para eliminar pregunta si se desea eliminar.

- Recibe el índice del registro y la razón de la consulta.

```
void PrintRegReg(int indice,int razon)
{
    FILE *fa;
    TWrkr reg;
    int del;
    fa=fopen("datos.dat","rb+");
    if(fa)
    {
        fseek(fa,sizeof(TWrkr)*indice,SEEK_SET);
        if(fread(&reg,sizeof(TWrkr),1,fa)==1)
        {
            if(reg.status)
            {
                if(razon==ELIMINAR)
                {
                    printf("\nNumero de Empleado: %d\n",reg.key);
                    printf("Nombre: %s\n",reg.name);
                    printf("Apellido Paterno: %s\n",reg.apPat);
                    printf("Apellido Materno %s\n",reg.apMat);
                    printf("Sexo: %s\n",reg.sexo);
                    printf("Pueso: %s\n",reg.puesto);
                    printf("Estado: %s\n",reg.estado);
                    printf("Edad: %d\n",reg.edad);
                    printf("Telefono: %d\n",reg.telefono);
                    del=validNum("Deseas eliminar al empleado? (0:Si 1:N
                    if(!del)
                    {
                        fseek(fa,-sizeof(TWrkr),SEEK_CUR);
                        fseek(fa,offsetof(TWrkr,status),SEEK_CUR);
                        fwrite(&del,sizeof(int),1,fa);
                        printf("\nSe elimino el empleado\n");
```

Ordenación

Función OrdenarIndex:

En la función menú, si la bandera de orden es 0 se manda a llamar esta función.

Ordena el arreglo de índices según el método apropiado.

- Recibe el arreglo de índices, el numero de empleados activos y el método de ordenación que será usado.

Los métodos usados son el de la burbuja si tamaño del arreglo es menor a 3500.

De 3500 a 3750 se usa HeapSort y de 3750 en adelante se usa MergeSort.

```
void OrdenarIndex(Tindex vect[],int i,int MetodoOrden)
{
    switch(MetodoOrden)
    {
        case 0:
            BubbleSortIndex(vect,i);
        case 1:
            HeapSortIndex(vect,i);
        case 2:
            MergeSortIndex(vect,0,i-1);
            break;
    }
}
```

Mostrar Todo

Función PrintRegTab:

Se pregunta de que forma se quiere mostrar los registros, normal, lo lee directo del archivo binario de registros, ordenado, si es necesario ordena el arreglo de índices e imprime haciendo consultas al archivo binario a partir del arreglo de índices ordenado.

- Recibe el arreglo de índices, el número de empleados, la dirección de memoria de la bandera de orden y el método de ordenación apropiado.

```
void PrintRegTab(Tindex vect[],int i,int &orden,int MetodoOrden)
{
    int op;
    printf("Mostrar Registros\n\n");
    printf("1) Normal\n");
    printf("2) Ordenado\n");
    printf("3) Regresar\n");
    op=validNum("Escoge una opcion: ",1,3);

    switch(op)
    {
        case 1:
            printf("\n");
            PrintArchBin();
            break;
        case 2:
            if(orden)
            {
                printf("\n");
                PrintBinIndex(vect,i);
            }
            else
            {
                printf("\n");
                OrdenarIndex(vect,i,MetodoOrden);
                orden=1;
                PrintBinIndex(vect,i);
            }
            break;
```

Función PrintArchBin:

Lee registro por registro cargando desde el archivo binario y muestra los datos del registro cargado si esta activo.

```
void PrintArchBin(void)
{
    FILE *fa;
    TWrkr reg;
    int k=0;
    fa=fopen("datos.dat","rb");
    if(fa)
    {
        printf("%s\n",ENCABEZADO);
        while(fread(&reg,sizeof(TWrkr),1,fa)==1)
        {
            if(reg.status)
            {
                printf(" %-4d ",++k);
                //getch();
                printf("| %6d | %-22s | %-16s | %-17s | %-20s | 646
            }
        }
        fclose(fa);
    }
    else
    {
        printf("\nEl archivo de registros no existe\n");
    }
}
```

Función PrintBinIndex:

Realiza consultas al archivo binario a partir del arreglo de índices ordenado e imprime a los empleados activos.

- Recibe el arreglo de índices, y el numero de empleados.

```
void PrintBinIndex(Tindex vect[],int i)
{
    FILE *fa;
    TWrkr reg;
    int j,k=0;
    fa=fopen("datos.dat","rb");
    if(fa)
    {
        for(j=0;j<i;j++)
        {
            fseek(fa,0,SEEK_SET);
            fseek(fa,sizeof(TWrkr)*vect[j].indice,SEEK_SET);
            if(fread(&reg,sizeof(TWrkr),1,fa)==1)
            {
                if(reg.status)
                {
                    printf(" %-4d ",++k);
                    printf("| %6d | %-22s | %-16s | %-17s |
                }
            }
        }
        fclose(fa);
    }
    else
    {
        printf("\nEl archivo de registros no existe\n");
    }
```

Generar archivo de texto

Función GenArchivoTxt:

Pregunta el nombre del archivo, no se pueden generar dos archivos con el mismo nombre.

Igual que en la función de impresión, pregunta si se generara normal u ordenado, si es normal lo lee del archivo binario y lo guarda en un archivo de texto, si es ordenado realiza consultas a partir del arreglo de índices ordenado y lo agrega al archivo de texto.

- Recibe el arreglo de índices, el número de empleados, la dirección de memoria de la bandera de orden y el método de ordenación apropiado.

```
switch(op)
{
    case 1:
        printf("\n");
        GenArchTxtNormal(name);
        printf("\nArchivo %s\n\n",name);
        PrintArchTxt(name);
        break;
    case 2:
        if(orden)
        {
            printf("\n");
            GenArchTxtOrden(vect,i,name);
            PrintArchTxt(name);
        }
        else
        {
            printf("\n");
            OrdenarIndex(vect,i,MetodoOrden);
            orden=1;
            GenArchTxtOrden(vect,i,name);
            PrintArchTxt(name);
        }
        break;
```

Función GenArchTxtNormal:

Funcionan igual a las funciones PrintArchBin y PrintBinIndex para imprimir todos los registros, pero en lugar de mostrarlos al usuario los escribe en el archivo de texto.

```
void GenArchTxtNormal(char name[])
{
    FILE *bin,*txt;
    TWrkr reg;
    int k=0;
    bin=fopen("datos.dat","rb");
    if(bin)
    {
        txt=fopen(name,"w");
        fprintf(txt,"%s\n",ENCABEZADO);
        while(fread(&reg,sizeof(TWrkr),1,bin)==1)
        {
            if(reg.status)
            {
                fprintf(txt,"| %-4d | %6d | %-22s | %-16s | %-17s | %-
            }
        }
        fclose(txt);
        fclose(bin);
    }
```

```
void GenArchTxtOrden(Tindex vect[],int i,char name[])
{
    FILE *bin,*txt;
    TWrkr reg;
    int j,k=0;
    bin=fopen("datos.dat","rb");
    if(bin)
    {
        txt=fopen(name,"w");
        fprintf(txt,"%s\n",ENCABEZADO);
        for(j=0;j<i;j++)
        {
            fseek(bin,0,SEEK_SET);
            fseek(bin,sizeof(TWrkr)*vect[j].indice,SEEK_SET);
            if(fread(&reg,sizeof(TWrkr),1,bin)==1)
            {
                if(reg.status)
                {
                    fprintf(txt," %-4d ",++k);
                    fprintf(txt,"| %6d | %-22s | %-16s | %-17s | %-20s | 6
                }
            }
        }
        fclose(txt);
        fclose(bin);
```

Empaquetar

Función empaquetar:

Limpia al archivo binario de registros principal los empleados inactivos.

Renombra el archivo binario principal para que sea temporal, del archivo temporal asigna según corresponda, al archivo de respaldo lo copia tal cual, al nuevo archivo principal asigna los empleados activos y al archivo de empleados eliminados asigna los inactivos.

- Recibe el arreglo de índices, el número de empleados, y cuantos respaldos se han hecho.

```
void empaquetar(Tindex vect[],int i,int resplado)
{
    FILE *bin,*binbak,*bindel,*tmp;
    TWrkr reg;
    char binBakName[30];
    char c[2];
    bin=fopen("datos.dat","rb");
    if(bin)
    {
        fclose(bin);
        rename("datos.dat","datos.tmp");
        tmp=fopen("datos.tmp","rb");
        if(tmp)
        {
            if(resplado==0)
            {
                bin=fopen("datos.dat","wb");
                binbak=fopen("datos.bak","wb");
                bindel=fopen("datosdel.bak","ab");
                //respaldo
                while(fread(&reg,sizeof(TWrkr),1,tmp))
                {
                    if(reg.status)
                    {
                        fwrite(&reg,sizeof(TWrkr),1,bin);
                        fwrite(&reg,sizeof(TWrkr),1,binbak);
                    }
                    else
                    {
                        fwrite(&reg,sizeof(TWrkr),1,bindel);
                        fwrite(&reg,sizeof(TWrkr),1,binbak);
                    }
                }
```

El numero de respaldos hechos se usa para nunca perder los datos y guardarlos en un archivo llamado “datosN.bak”, donde N es el número de respaldo hecho.

```
if(resplado>0)
{
    strcpy(binBakName,"datos");
    strcat(binBakName,itoa(resplado,c,10));
    strcat(binBakName,".bak");
    bin=fopen("datos.dat","wb");
    binbak=fopen(binBakName,"wb");
    bindel=fopen("datosdel.bak","a");
    //respaldo
    while(fread(&reg,sizeof(TWrkr),1,tmp))
    {
        if(reg.status)
        {
            fwrite(&reg,sizeof(TWrkr),1,bin);
            fwrite(&reg,sizeof(TWrkr),1,binbak);
        }
        else
        {//332686
            fwrite(&reg,sizeof(TWrkr),1,bindel);
            fwrite(&reg,sizeof(TWrkr),1,binbak);
        }
    }
    printf("\nSe empaqueto correctamente\n");
    fclose(tmp);
    fclose(bindel);
    fclose(binbak);
    fclose(bin);
}
```

Conclusión:

Al realizar la practica note que el uso de archivos binarios indexados puede reducir un poco el rendimiento del programa, pero esto a cambio de reducir la carga en la memoria RAM permitiendo tener un numero mayor de registros, también se debe tener en cuenta que el archivo se lea con “modob” indicando el archivo binario, ya que este fue mi principal error al realizar la actividad, olvidar indicar el modo binario, también el uso de archivos de texto para guardar los datos de un archivo binario es un buen método de comprobar las salidas ya que al trabajar con tantos datos impresos pueden ocurrir errores al mostrarlos y dificulta la depuración del código.

6. Anexos

PDF:

https://drive.google.com/file/d/1U3tRDnvnlvDEXb3HMlkNsUSiIi4pj4FH/view?usp=sharing

Repositorio:

https://github.com/Muners24/Programacion-Estructurada

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

El Lenguaje de programación C (S/f). Universidad de Coruña. Recuperado el 20 de noviembre de 2023, de https://www.dc.fi.udc.es/~so-grado/current/Varios/CursoC.pdf