Unidad 10 Estructuras de Almacenamiento físico

10

Contenidos analíticos

Parte IV Almacenamiento Físico - Estructuras con asignación dinámica
Estructuras de Almacenamiento físico

Concepto de almacenamiento fisico. Archivos de texto y archivos binarios. Archivos con acceso directo. Analisis de bibliotecas para el manejo de archivo. Nombre lógico y fisico.
Patrones algoritmicos: comparación con patrones de otras estructuras de manejos de conjunto de datos del mismo tipo, conservando la lógica y modificando el modo de acceso a cada uno de los datos particulares

Estructura tipo Archivo

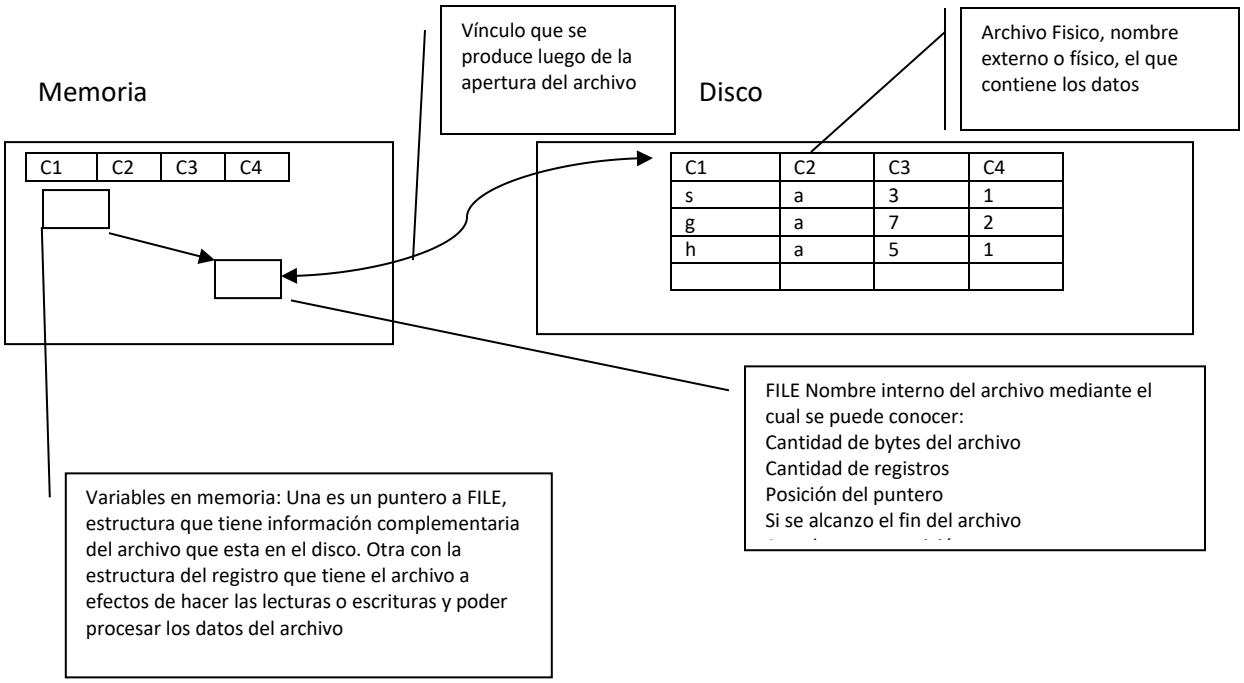
- Archivos
  - De texto
  - Binarios
    - De tipo
      - De tipo registroCon registros de tamaño fijo
    - Con acceso directo
    - En la implementación en C utilizamos
      - FILE *
      - fopen
      - fread
      - fwrite
      - feof
      - fseek
      - ftell
      - fclose

.FILE * F = fopen(“nombre externo “, “modo de apertura”); se establece una línea de comunicación con el archivo.

Modo	Descripción
r	Reset Abre archivo de texto para lectura
rt	Idem anterior, explicitando t: texto
w	Write Abre archivo de texto para escritura, si el archivo existe se descarta el contenido sin advertencia
wt	Idem anterior, explicitando t: texto
rb	Reset abre archivo binario para lectura
wb	Write Abre archivo binario para escritura, si el archivo existe se descarta el contenido sin advertencia
+	Agrega la otra modalidad a la de apertura

Para controlar que la apertura haya sido correcta se puede:
If ((F = fopen(“Alumnos”, “wb+”)) == NULL) {error(1); return 0};
Si el apuntador es NULL, el archive no se pudo abrir.

FILE* f=fopen(“SL”,” “)



Archivos binarios:

Análisis comparativo entre arreglos y archivos de registro de tamaño fijo		
Propiedad	Arreglo	Archivo
Tamaño Físico en Tiempo Ejecución	Fijo	Variable
Almacenamiento	Electrónico	Físico
- Persistencia	No	Si
- Procesamiento	Rápido	Lento
Búsquedas		
- Directa	Si	Si
- Binaria	Si (si esta ordenado)	Si
- Secuencial	Es posible	No recomendada
Carga		
- Secuencial	Si	Si (al final de la misma)
- Directa	Si	Solo con PUP
- Sin repetir clave	Si	No recomendada
Recorrido		
- 0..N	Si	Si
- N..0	Si	Si
- Con corte de control	Si	Si
- Con Apareo	Si	Si
- Cargando los N mejores	Si	No recomendada
Ordenamientos		
- Con PUP	Si	Si
- Método de ordenamiento	Si	No recomendado

Definiciones y declaraciones:

Archivo binario

Numero	Cadena	Caracter
int	char cadena[N]	char

```
struct   TipoRegistro {
         int Numero;
         char Cadena[30];
         char C;
} Registro;
FILE * F;
```

Operaciones simples

```
FILE * abrirBinLectura( FILE * f, char nombre[]){
// Asocia el flujo f  con el archivo nombre, lo abre en modo lectura
      return fopen(nombre, "rb");
}
FILE *abrirBinEscritura( FILE * f, char nombre[]){
// Asocia el flujo f  con el archivo nombre, lo abre en modo escritura
      return fopen(nombre, "wb");
}
```

```
int  cantidadRegistros( FILE * f){
// retorna la cantidad de registros de un archivo
	TipoRegistro r;
	int posActual = ftell(f), cantReg;
	fseek(f, 0, SEEK_END); //pone al puntero al final del archivo
	cantReg = ftell(f)/sizeof(r); //cantidad de bytes desde el principio/tamaño del registro
	fseek(f, posActual, SEEK_SET);//vuelve a posición inicial
	return cantReg;
}

int  posicionPuntero( FILE * f){
// retorna el desplazamiento en registros desde el inicio
	TipoRegistro r;
	return ftell(f)/sizeof(r); //cantidad de bytes desde el principio/tamaño del registro
}

int  seek( FILE * f, int pos){
// Ubica el puntero en el registro pos (pos registros desplazados del inicio)
	TipoRegistro r;
	return fseek(f, pos * sizeof(r), SEEK_SET);
}

int leer( FILE * f, TipoRegistro *r){
	//lee un bloque de un registro, retorna 1 si lee o EOF en caso de no poder leer.
	return fread(&r, sizeof(r) , 1, f);
}

int grabar( FILE * f, TipoRegistro r){
	//Graba un bloque de un registro.
	return fwrite(&r, sizeof(r) , 1, f);
}

int leerArchivoCompletoV1(FILE * f){
	// registro por registro con lectura anticipada
	Tr r;
	fread(&r, sizeof(r) , 1, f);
	while(!feof(f)){
		//procesar r;
		fread(&r, sizeof(r) , 1, f);
	}
	return 0;
 }
```

```
int leerArchivoCompletoV2(FILE * f){
        // registro por registro con lectura y control simultaneo
        Tr r;
        while(fread(&r, sizeof(r) , 1, f){
                //procesar r;
        }
 }

int LeerArchivoCompleto( FILE * f, TipoVector v[], int N){
        //lee un archivo y los guarda en un vector (el tamaño debe conocerse a priori).
        return fread(v, sizeof(v[0]) , CantidadRegistros(f), f);
}
```

EJEMPLO CON REPRESENTACION GRAFICA

Archivos binarios: Analisis, síntesis y comparaciones

En la implementación en C utilizamos

FILE* se asocia a un flujo para controlarlo→ FILE* f;

fopen asocia el nombre lógico al físico→ f=fopen(“NombreFisico”,”rb+”);

fread lectura por bloques→fread(&r, sizeof(r), 1, f);

fwrite escritura por bloques→fwite(&r, sizeof(r), 1, f);

feof indicador de final del archivo→feof(f)

fseek permite acceso directo→ fseek(f,2*sizeof(r),SEEK_SET)

ftell retorna de bytes de desplazamiento desde el inicio del flujo→ftell(f);

fclose cierra el archivo, vacía el buffer y coloca marca de EOF→flose(f)

struct tr{ int c1; int c2;} → supongo entero de 8 bytes

8		16		24		32		40	
3	5	4	1	2	8	14	5	5	11

ftell(f) →8 24 36

fseek(f, 16, SEEK_SET) →

fseek(f, 2*8, SEEK_CUR) →

fseek(f, -sizeof(r), SEEK_END) ←

ftell(f)/sizeof(r) → 2 → 4

fseek(f,0,SEEK_END); p = ftell(f)/sizeof(r);→calcula cantidad de registros

tr v[10]

FILE* f = fopen(“ma”,”rb+”)

a=fread(&reg, sizeof(reg),1,f)

3	5

fread(&a, sizeof(int),1,f)

3

fread(v, sizeof(reg),4,f) v → vector de struct; v[0] → struct pos 0

fread(v, sizeof(reg),4,f)
fread(&v[1], sizeof(reg),4,f)

```
fread(    );
while(! feof(f) ){
 insertarordenado(lista, reg);
fread(    );
}
```

```
while(fread(   )){
insertarordenado(lista, reg);
}
```

Análisis comparativo Archivos Vectores→Declaración, Acceso, busqueda

Vector (de registros)	Archivo (de registros)
Consideraciones generales	
Almacenamiento lógico (memoria) Tamaño Fijo (T. E.). Procesamiento rápido Sin persistencia después aplicación Prioriza velocidad procesamiento	Almacenamiento físico (disco) Tamaño variable (T.E.). Procesamiento lento Con persistencia después aplicación Garantiza persistencia
Definiciones y declaraciones	
Tr V[X]; Note que determina a priori el tamaño (X) y especifica el tipo de dato de cada posición(Tr).	FILE* f = fopen (“XXXX”, “rb+”); Solo indica como lo abre sin especificar particularidad del dato, solo “b o t” y no el tamaño
Acceder al registro de posición N	

V[N];
Accede al registro en memoria con esa posición.
	fseek(f, N*sizeof(r), SEEK_SET);
	posiciona el puntero en el registro
	fread(&r, sizeof(r), 1, f);
	Lleva a memoria el registro N.

Modificar el registro de la posición N	

V[N].campo = valor
Modifica el campo especifico del registro N que está en memoria
	fseek(f, N*sizeof(r), SEEK_SET);
	APUNTAR al registro a modificar
	fread(&r, sizeof(r), 1, f);
	LEER Lo lleva a memoria.
	r.campo = valor
	MODIFICAR el dato en memoria
	fseek(f, N*sizeof(r), SEEK_SET);
	Vuelve APUNTAR al registro.
	fwrite(&r, sizeof(r), 1, f);
	GRABAR en el disco.

Acceder al registro siguiente al de la posición N y tenerlo a disposición	

V[N+1]	fseek(f, (N+1)*sizeof(r), SEEK_SET);
	APUNTAR al registro a modificar
	fread(&r, sizeof(r), 1, f);
	LEER Lo lleva a memoria.

Acceder al primer registro y tenerlo a disposición	
V[0]	fseek(f, 0, SEEK_SET); fread(&r, sizeof(r), 1, f);

<table>
<tr><th colspan="2">Acceder al último registro y tenerlo a disposición</th></tr>
<tr><td>V[X-1]</td><td>fseek(f, -sizeof(r), SEEK_END);
fread(&r, sizeof(r), 1, f);</td></tr>
<tr><th colspan="2">Búsqueda binaria</th></tr>
<tr><td>int bb(Tr V[], int X, int N){
int p = 0;
int u = N-1;
int m;

while(p<=u){
 m = (p + u)/2;

if(V[m]. campo == X) return m;
 else if(X>V[m].campo p= m++;
  else u = m--;
}
return -1;
}</td><td>int bb(FILE* f,int X){
int p = 0;
int u = cantRegistros(f)-1;
int m;
Tr r;
while(p<=u){
 m = (p + u)/2;
 fseek(f,m*sizeof(r), SEEK_SET);
 fread(&r, sizeof(r), 1, f);
 if(r. campo == X) return m;
 else if(X>r.campo p= m++;
  else u = m--;
}
return -1;
}</td></tr>
<tr><th colspan="2">Búsqueda directa (PUP)</th></tr>
<tr><td>V[Clave-vInicial]</td><td>fseek(f,sizeof(r)*(Clave-vInicial),SEEK_SET)
APUNTAR al registro buscado
fread(&r, sizeof(r), 1, f);
LEER el registro apuntado</td></tr>
<tr><th colspan="2">Recorrido</th></tr>
<tr><td>i= 0;// ir al inicio

while(i<N) { //control de fin.
 procesar v[i];
 i++; //avanzar
 }// fin del ciclo</td><td>fopen pone epuntero al inicio
Opcion 1 lectura anticipada
fread(&r, sizeof(r), 1,f); // leer
while(!feof(f)){
  procesar r;
  fread(&r, sizeof(r), 1,f);
}
Opcion 2 lectura y verificacion simultaneal
while (fread(&r, sizeof(r), 1, f)){
  procesar r;
}</td></tr>
</table>

<table>
<tr><th colspan="2">Apareo (conceptual)</th></tr>
<tr><td>Concepto: Aplicable a dos o más estructuras son al menos un campo en común ordenadas por ese campo que se procesan paralelamente, intercalando los valores para conservar el orden</td><td>Seudocódigo:
Situarse al principio de ambas estructuras
Hacer mientras (haya datos en ambas)
Si la primera cumple criterio de ordenamiento
Procesarla;
Avanzar con ella
Sino
Procesar la otra;
Avanzar con ella
Fin del mientras
Agotar la estructura que no se termino
Procesarla;
Avanzar
Fin del proceso</td></tr>
<tr><th>Vectores</th><th>Archivos</th></tr>
<tr><td>int i = 0;
int j = 0;

while((i<N) && (j<M)){
if(v1[i].c1<v2[j].c1){
//procesar v1[i];
i++;}
else{
//procesar v2[j];
j++;}
}
while(i<N){
//procesar v1[i];
i++; }
while(j<M){
//procesar v2[j];
j++;}</td><td>fread(&r1,sizeof(r1), 1, f1);
fread(&r2,sizeof(r2), 1, f2);

while(!feof(f1)&& !feof(f2)){
if(r1.c<r2.c){
// procesar r1;
fread(&r1,sizeof(r1), 1, f1)};
else{
// procesar r2;
fread(&r2,sizeof(r2), 1, f2) ;}
};
while(!feof(f1)){
procesar r1;
fread(&r1,sizeof(r1), 1, f1)};
while(!feof(f2)){
procesar r2;
fread(&r2,sizeof(r2), 1, f2)};</td></tr>
</table>

1
8
14
22
125

2
3
11

```
int j = 0; int i = 0;
// j==M || i<N && v1[i].c < v2[j].c
while((i<N) || (j<M)){
                if(j==M || i<N && v1[i].c < v2[j].c){
                   //procesar v1[i];
                   i++;}
                else{
                   //procesar v2[j];
                   j++;}
```

Con plantilla y función criterio

Implementación de la función

```
template <typename T>
void Apareo(T v1[], T v2[], int N, int M, int
(*criterio)(T,T)){
   int i = 0;
   int j = 0;
   while((i<N) && (j<M)){
      if(criterio(v1[i],v2[j])==1){
         //procesar v1<T>[i];
         cout << v1[i].c1 << endl;
         i++;} else{
         //procesar v2<T>[j];
         cout << v2[j].c1 << endl;
         j++;}   };
while(i<N){
      //procesar v1<T>[i];
      cout << v1[i].c1 << endl;
      i++; }
while(j<M){
     //procesar v2<T>[j];
     cout << v2[j].c1 << endl;
     j++;  } }
```

programa principal e Invocación

```
struct tr{ int c1; intc2};
int criterioCampo1Asc(tr a,tr b){
  if(a.c1<b.c1) return 1;
  return 0;
}

int criterioCampo1Des(tr a,tr b){
  if(a.c1>b.c1) return 1;
  return 0;
}
int criterioCampo1yCampo2Des(tr a,tr b){
if(a.c1>b.c1||( a.c1==b.c1&& a.c2>b.c2) return 1;
  return 0;
}
int main (){
……….
Apareo<tr>(v1,v2,N,M,criterioCampo1Asc)

Apareo<tr>(v1,v2,N,M,criterioCampo1Desc)

Apareo<tr>(v1,v2N,M,,criterioCampo1yCampo2Desc)
```

Corte de Control (conceptual)

Concepto: Aplicable a una estructura con al menos un campo que se repite, agrupados por ese campo.
Propósito: Mostrar los datos en forma ordenada sin repetir ese campo común.
Como se requieren los datos

Materia	Legajo	Nota
AyED	145234	5
AyED	234169	3
AyED	135321	8
SSL	242132	9
SSL	125328	7
SSL	146342	2

Seudocódigo:

Situarse al principio de la estructura
Inicializar contadores generales
Mostrar títulos generales

Hacer mientras (haya datos)
  Inicializar contadores de cada subgrupo
  Mostrar títulos subgrupos
  Identificar grupo a analizar→subgrupo
  Hacer mientras (haya datos Y mismo subgrupo)
    Procesar el registro
    Avanzar al siguiente
  Fin ciclo interno
 Informar datos de cada subgrupo
Fin ciclo externo
Informar sobre generalidades

Materia AyED Orden Legajo Nota 1 145234 5 2 234169 3 3 135321 8 Cantidad Alumnos 3 Materia SSL Orden Legajo Nota 1 242132 9 2 125328 7 3 135321 2 Cantidad Alumnos 3 Total Alumnos 6	El ciclo interno tiene la conjunción de la condición del ciclo externo y la propia. Las lecturas deben hacerse al final del ciclo interno menor. Este Patrón algorítmico NO ORDENA, solo muestra agrupados los datos que ya están ordenados evitando repeticiones innecesarias.

Vectores	Archivos

int i = 0;
totalAlumnos = 0;
<TITULOS>
while(i<N){
alumnosCurso = 0
Control = V[i].curso
<TITULOS>
while(i>N&&V[i].curso==control)
alumnosCurso++
totalAlumnos ++
Mostrar Datos del V[i]
i++ }// fin ciclo interno
Mostrar Datos del Subgrupo
} // fin ciclo externo
Resultados Finales
	fread(&r1,sizeof(r1), 1, f1);
	totalAlumnos = 0;
	<TITULOS>
	while(!feof(f){
	alumnosCurso = 0
	Control = r.curso
	<TITULOS>
	while(!feof(f)&&control==r.curso)
	alumnosCurso++
	totalAlumnos ++
	Mostrar Datos del registro
	fread(&r1,sizeof(r1), 1, f1);}
	Mostrar Datos del Subgrupo
	} // fin ciclo externo
	Resultados Finales

Otros patrones Apareo

Apareo por criterios diferentes

- Por más de un campo de ordenamiento
  - conservar algoritmia modificando la lógica → agrupar criterios →&&
- Modificar criterio
  - Cambiando algoritmia o Utilización de funciones de criterio

Apareo de N estructuras

- Ir apareando por pares y luego aparear loe resultados
- Utilizar una estructura auxiliar →V1

A1

4
6
12
20

A2

3
7
19
25

A3

1
2
5
8

V1

Clave	Control
20	0
25	0
8	0

```
fread(&r1,sizeof(r1), 1, f1)
V1[0].clave = r1.clave
fread(&r2,sizeof(r2), 1, f2)
V1[1].clave = r2.clave
fread(&r3,sizeof(r3), 1, f3)
V1[2].clave = r3.clave
archivosActivos = 3
while(archivosActivos>0){
  p = buscarMinimoNoCero(V)
switch(p)
{
   case 0: //procesar archivo1, si es feof poner 0 en el control y disminuir archivosActivos
   break;
   case 1: // procesar archivo2, si es feof poner 0 en el control y disminuir archivosActivos;
   break;
   case 2: // procesar archivo3, si es feof poner 0 en el control y disminuir archivosActivos
   break;
}
```

```
int buscarMinimoNoCero( tr V[], int N){
int i, minimo;
for( i = 0;V[i].control!=0; i++);
minimo = V[i].clave; i++;
for( ; i<N ; i++){
 if(V[i].clave < minimo) minimo = V[i].clave;
……..}
return posición del minimo
```

4	Juan
2	Jose
6	Pedro
8	Pablo
9	Ana
1	maria

Como conocemos el tamaño guardamos en un vector

```
struct tr {
int NL;
char Nombre[20];
};
Tr R; int i;
Tr Vector[10];
FILE* f = fopen (“Alumnos”,  “rb+);
fread(Vector,sizeof(R),6,f);//archivo y vector ambos desordenados
OrdenarVector(Vector,6);//archivo desordenado, vector ordenado
```

1	maria
2	Jose
4	Juan
6	Pedro
8	Pablo
9	Ana

```
// mostrar los datos por pantalla orenados→ datos que están en el vector
for( i=0;i<6;i++)
        cout<<Vector[i].NL << Vector[i].Nombre;


//ordenar el archivo → usando vector estructura auxiliar
fseek(f,0,SEEKSET);
fwrite(Vector,sizeof(R),6,f);//guarda todos los datos del vector en el archovo
```

4	Juan
2	Jose
6	Pedro
8	Pablo
9	Ana
1	maria

Como NO conocemos el tamaño guardamos en una lista

```
struct tr {
int NL;
char Nombre[20];
};
struc Nodo{
 tr info;
Nodo* sgte
};
Nodo* lista = NULL;
Nodo * P = NULL
```

Info		Sgte
NL	Nombre	Nodo *

```
Tr R; int i;
FILE* f = fopen ("Alumnos",  "rb+);


while(fread(&R,sizeof(R),1,f)//cargar archivo en lista. Archivo desordenado, lista ordenada
        InsertarOrdenado(lista,R);

// mostrar los datos por pantalla ordenados→ datos que están en la lista
// recorro la lista sin vaciarla
P = lista
for( ;P!=NULL;){
        cout<<P->info.NL <<P->info,Nombre;
        P= P->sgte;
};

//ordenar el archivo → usando lista estructura auxiliar y vacindola
fseek(f,0,SEEKSET);

while(lista!=NULL)
        R = pop(lista);
        fwrite(&R,sizeof(R),1,f);//guarda todos los datos del vector en el archivo
```

NL Nombre

1	R
4	D
5	W

NL Nombre

2	J
3	A
7	M
9	G
14	P
25	Q

Archivo Leer(A1, R1)

R1

Fin de archivo	W

R2

9	G

```
fread(&R1, sizeof(R1), 1, A1);
fread(&R2, sizeof(R1), 1, A2);
while(!feof(A1)&&!feof(A2)){ // hay datos en ambos
        if(R1.NL < R2.NL){
                cout<<R1.NL<<R1.nombre;
                fread(&R1, sizeof(R1), 1, A1);
                }
        else{
                cout<<R2.NL<<R2.nombre;
                fread(&R2, sizeof(R2), 1, A2);
                }
};

while(!feof(A2)){ // agota el 2do si no se termino
        cout<<R2.NL<<R2.nombre;
        fread(&R2, sizeof(R2), 1, A2);
}

while(!feof(A1)){ //agota el 1ro I no se termino
        cout<<R1.NL<<R1.nombre;
        fread(&R1, sizeof(R1), 1, A1);
}
```

1	
2	
4	

Ejercicio

la universidad dispone de un archivo de alumnos “anterior.dat”, cada registro con

Legajo	codigoMateria	FechaInscripcion	otros
Entero	Entero	entero	cadena 20 caracteres

El archivo esta ordenado por legajo y código de materia con todas las inscripciones anteriores de los alumnos.

Ademas deispone de in vector inscripcionesDelDia, del mismo tipo de registro, sin orden con N componentes [1..500]

Se pide desarrollar un programa que actualice el archivo “anterior.dat” con las inscripciones del dia que se encuentran en el vector, generando el archivo “actualizado.dat” con el mismo criterio de ordenamiento que anterior . dat

Nota: las estructuras dato ya están cargadas, no registran errores de ningún tipo

Consultas extras.

1. si ambas estructuras dato, hubieran estado desordenada.
    a. podría generar el archivo actualizado ordenado
    b. justifique su respuesta en caso de ser negativa
    c. plantee su estrategia en caso de creer que es posible
2. como resolvería la situación si el vector estuviera ordenado por legajo
3. se puede renombrar el archivo actializado.dat como anterior.dat

Operaciones sobre archivos implementadas en C, C++

El subconjunto de funciones de C, declaradas en el archico cabecera de entradad y salida que utilizaremos son:

`fopen` - Abre un archivo.
`fwrite` - Graba datos en el archivo.
`fread` - Lee datos desde el archivo.
`feof` - Indica si quedan o no más datos para ser leidos desde el archivo.
`fseek` - Permite reubicar el indicador de posición del archivo.
`ftell` - Indica el número de byte al que está apuntando el indicador de posición del archivo.
`fclose` - Cierra el archivo.

Utilizando las funciones anteriores y con el propósito de facilitar las implementaciones, en hojas anteriores se han desarrollado las siguientes funciones propias, que utilizaremos, en algunos casos, a efectos de facilitar la comprensión, estas son:

`seek` – Posiciona el puntero en una posición dada.
`cantidadRegistros` – Indica cuantos registros tiene un archivo.
`posicionPuntero` – Retorna el desplazamiento en registros desde el inicio.
`leer` – Lee un registro del archivo.
`grabar` – Graba un registro en el archivo.

Grabar un archivo de caracteres

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main()
{
      // abro el archivo; si no existe => lo creo vacio
      FILE* arch = fopen("DEMO.DAT","wb+");
      char c = 'A';
      fwrite(&c,sizeof(char),1,arch); // grabo el caracter 'A' contenido en c
      c = 'B'; // C provee también fputc(c, arch);
      fwrite(&c,sizeof(char),1,arch); // grabo el caracter 'B' contenido en c
      c = 'C';
      fwrite(&c,sizeof(char),1,arch); // grabo el caracter 'C' contenido en c
      fclose(arch);
  return 0;}
```

Leer un archivo caracter a caracter

```
#include <iostream>
#include <stdio.h>
using namespace std;
int main() {
      FILE* arch = fopen("DEMO.DAT","rb+");
      int c;
      fread(&c,sizeof(char),1,arch);
      while( !feof(arch) ){
        cout << c << endl;
        fread(&c,sizeof(char),1,arch);
      }
      fclose(arch);
  return 0;
}
```

Archivos de registros

```
struct Alumno
{
      int dni;
      char nombre[25];
};
```

Grabar un archivo de registros

Lectura de datos por teclado y se graban los mismos en un archivo.

```
#include <iostream> //la inclusion de las cabeceras necesarias
#include <stdio.h>
#include <string.h>
using namespace std; //declaracion del espacio de nombre
int main()
{
      FILE* f = fopen("Alumnos.DAT","w+b");
      int dni;
      string nom;
      Alumno a;
      // ingreso de datos
      cout << "Ingrese dni";
      cin >> dni;
      while( dni>0 ){
        cout << "Ingrese nombre";
         cin >> dni;
         a.dni = dni;
         strcpy(a.nombre,nom.c_str()); //
         fwrite(&a,sizeof(Alumno),1,f); // grabo la estructura en el archivo
         cout << "Ingrese dni: ";
         cin >> dni;
        }
      fclose(f);
  return 0;
}
```

Leer un archivo de registros

Mostrar el contenido del archivo cargado en el punto anteriorA continuación veremos un programa que muestra por consola todos los registros del archivo `PERSONAS.DAT`.

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int main()
{
      FILE* f = fopen("Alumnos.DAT","rb+");
      Alumno a;

      while(fread(&a,sizeof(Alumno),1,f)){
        cout << p.dni << ", " << p.nombre <<", " << p.altura << endl;
            }
      fclose(f);
 return 0; }
```

Acceso directo a los registros de un archivo

Acceder al registro de la quinta posición y mostrar su contenido

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int main()
{
      int N=10;
      FILE* f = fopen("Alumnos.DAT","r+b");
      Alumno a;

      fseek(f,sizeof(Alumno)*(N-1), SEEK_SET);
      // se ubica el puntero al comienzo del enesimo registro
      fread(&a,sizeof(a),1,f); //sizeof puede ser de tipo o de variable
      // muestro cada campo de la estructura leida
      cout << p.dni << ", " << p.nombre <<endl;
  return 0;
}
```

Acceder al ultimo registro mostrar su contenido

```
#include <iostream>
#include <stdio.h>
#include <string.h>
using namespace std;
int main()
{
      FILE* f = fopen("Alumnos.DAT","r+b");
      Alumno a;

      fseek(f,-sizeof(Alumno), SEEK_END);
      // se ubica el puntero al comienzo del ultimo registro
      fread(&a,sizeof(a),1,f); //sizeof puede ser de tipo o de variable
      // muestro cada campo de la estructura leida
      cout << p.dni << ", " << p.nombre <<endl;
  return 0;
}
```

Templates

Template: leer

```
template <typename T> T leer(FILE* f)
{
	T R;
	fread(&R,sizeof(T),1,f);
	return R;
}
```

Template: grabar

```
template <typename T> void grabar(FILE* f, T R)
{
	fwrite(&R,sizeof(T),1,f);
	return;
}
```

Template: seek

```
template <typename T> void IrA(FILE* arch, int n)
{
	// SEEK_SET indica que la posicion n es absoluta respecto del inicio del archivo
	fseek(arch, n*sizeof(T),SEEK_SET);
}
```

Ejemplos

Leer un archivo de registros usando el template `leer`.

```
	f = fopen("Alumnos.DAT","rb+");
	// leo el primer registro
	T R;
	while( leer<Alumno> (f) ){

	  cout << R.dni<<", "<<R.nombre << endl;

	}
	fclose(f);
```