Unidad 13 Combinaciones complejas de estructuras

13

Contenidos analíticos

Parte IV Almacenamiento Físico - Estructuras con asignación dinámica
Combinaciones complejas de estructuras

Resolución de problemas complejos combinando las estructuras analizadas y justificando la elección de las mismas.

Retomando lo visto a lo largo del año en Algoritmos

Sentecias simple
Asignación
- Interna
- cin
- cout

sentencias estructuradas
- if
- while
- for

{ }
Incluir bibliotecas, espacios de nombre
Funciones
- Declaración
- Definición
- Argumentos→volorL
- Parámetros →valor, referencia

Reusabilidad
Generalidad
Tipo valor de retorno
Tipo de dato
- Simples
- Punteros
- Estructuras

definicion
registro . ->
patrones
- carga → directa, secuencial, sin repeticiones
- recorrido → totalmente, parte, corte de control, apareo
- búsqueda → directa, secuencial, dicotómica
- ordenamiento → método vector, insertar ordenado PUP

ordenar archivo
PUP clave numérica, posicional
Cargar en vector, ordenar vector volcar el vector a un archivo

```
I=0;                    lista = NULL;
while(fread(&r,   ){    ==
    v[i]=r;             insertarOrdenado(Lista,r);
    i++
}                       }
ordenarVector(v,n)
fseek(f,0,SEEK_SET)     =
for(i=0;i<n; i++)       while(lista)
    r=v[i]                  r=pop(lista)
    fwrite(------)          =
}
fclose                      =
```

Cargar en lista ordenada volcar los datos de la lista al archivo

array

archivo
registro de tamaño fijo → búsqueda directa
FILE *
fopen
fread
fwrite
ftell sizeof
fseek
fclose
borrar, renombrar

enlazadas
pila,
void push(Nodo* & p, td x)
td pop(Nodo*&p)
cola
queue
unqueue
lista
Nodo* insertarOrdenado(Nodo *&lista, td x)
Buscar
Cargarsinretir
Acumula por una clave
Lista de listas

Combinación de estructuras
struct cada campo puede tener: dato simple, puntero, vector
array en cada posición dato simple struct
lista info puede ser: dato simple, vector, struct
archivos binarios: datos simples, struct

vector/matriz de punteros

vector/de registros
int v[5] v[3]
int m2[5][5] m2[2][3]
int m3[5][5][5] m3[2][1][3]
vector de listas
matriz de listas
lista ordenada por dos campos
lista de listas

insertar ordenado(lista tr x) Cambiar logica y conservar algoritmia
primer punto de selección → menor 1ra posicion
1 campo creciente if(lista == NULL || x.c1<lista->info.c1)
1 campo de creciente if(lista == NULL || x.c1>lista->info.c1)
2 campo creciente if(l == NULL ||(x.c1<l->info.c1||x.c1==l->info.c1&&x.c2<l->info.c2))
Segundo punto de selección → while en medio o al final
while(p->sgte!=NULL&& x.c1>p->sgte->info.c1)
para mas de un criterio similar a lo del primer punto

cuando en memoria no podemos guardar todos los datos de todos los registros del archivo
registro con muchos campos estoy restringido en memoria

pos	Registro								
0	C1								
1									
2									
3									
4									
5									
6									

Ordenar por campo1 todo el registro la memoria restringida

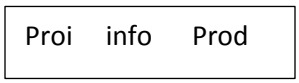
C1	refArchivo

```
FILE* Aux= fopen("      ",wb+);
TipoInfo X;
tipoRegistro R;

 Lista =NULL;
I=0;
while(fread(&r,……..,f){
        x.c1=r.c1;
        x.refArchivo = i;
        insertarOrdenado(lista, x);
        i++;
}

while(Lista) {
        x=pop(lista);
        fseek(f, x.refArchivo*sizeof(r),SEEK_SET);
        fread(&r,sizeof(r), 1,f);
        fwrite(&r,----------, 1, Aux);
}
```

Arboles
Orden



Binario de búsqueda

10 8 6 14 12 16 9

10 0
8 14 1
6 9 12 16 2

Balanceado
1 3 4 6 7 9 12

1
3
4
6
Vn Mi Vn 7
Común inversa
In vi MI vd
Pre Mi vi vd
Pos vi vd mi

6 8 9 10 12 14 16
Nivel profundidad rotación
Cargar un árbol binario de búsqueda balanceado con un int en la información, sin repetición del dato, en forma iterativa. Mostrar el dato con 1 de los recorridos a eleccion

Como pensar

Estructuras de datos: Características

	Vector	Archivo	Listas ordenadas
Almacenamiento	Logico	fisico	Logico
Procesamiento	Rapido	Lento	Rapido
Tamaño en T.E.	Fijo	Variable	Variable
Tamaño cada posicion	Informacion	Informacion	Info + sgte
Busqueda directa	SI→V[N]	Fseek	Buscar secuencialmente
Persistencia	NO	SI	NO
Busqueda binaria	SI	SI	NO→arbol

Busqueda secuencial	SI	NO se recomienda	SI→unica posible
Carga directa	SI →V[N] = valor	Acceder → fseek Grabar → frwrte	Utilizando diferentes funciones
Carga secuencial	SI→v[ultimaPos] = valor	Acces. a ultima pos Grabar	SI Insertar ordenado
Carga sin repetir	Buscar si no esta se agrega	NO se recomienda Est. Auxiliares	Buscar si no esta se agrega
Ordenamiento	PUP Metodos de ordenam.	PUP Solo usando est. aux	insertarOrdenado
Definicion	Td Nombre[tamaño]	FILE*f; f= fopen(.....)	Definir Nodo → struct Punt Nodo* X = NULL
Acceso	V[N]	Acceder → fseek leerr → fread	Buscar secuencialmente
Corte de control	SI	SI	SI
Apareo	SI	SI	SI
Modificar la pos N	V[N] = valor	Apuntar → fseek Leer→fread Modificar en memoria Apuntar → fseek Grabar → fwrite	Buscar secuencialmente modificar

Estructuras de datos: Criterios de selección

1. Priorizar de ser posible acceso directo y velocidad de procesamiento.
    a. Vector en primer lugar si se cumple tamaño fijo, razonable y conocido a priori, y sin necesidad de persistencia→ acceso directo, búsqueda binaria, búsqueda secuencial
2. Si no se conoce el tamaño y no se requiere persistencia
    a. Estructuras enlazadas
        i. Pila si se debe invertir el orden o si es irrelevante
        ii. Colas si se debe mantener
        iii. Listas si se debe generar
3. Archivo si se requiere persistencia: directamente o estructuras auxiliares después cargar al archivo

Estructuras → muchos datos del mismo tipo

Archivos
Vectores → memoria → mejorar tiempos de procesamiento → tamaño fijo
Listas ordenadas → memoria →mejorar tiempos de procesamiento → tamaño variable

Observen como vienen los datos:

1. Es posible que los datos de partida estén en la forma en que nos piden los resultados→ procesamos directamente el archivo
2. Es posible que los datos de partida NO estén en la forma en que nos piden los resultados→ Debemos cargar en estructuras auxiliares y procesar esta estructura, reordenar los datos del archivo y procesar el archivo

Datos provienen de archivos

1. Tal como están se muestran

Consigna →Mostrar todos los datos del archivo
No requiere guardarlo en otra estructura, leer cada registro y mostrarlo
Tr v[10], r; v 10
FILE* f = fopen("MiArchivo.dat","rb+) while(fread(&r,sizeof(r),1,f)) { cout<<r.c…..}
Leer todos los registros y ponerlos en un vector fread(v,sizeof(r),10,f)
Dato Archivo ordenado
Consigna →Mostrar todos los datos del archivo ordenados por el mismo campo
No requiere guardarlo en otra estructura, leer cada registro y mostrarlo
Dado un archivo de registros mostrar su contenido por pantalla
Los resultados están en el orden que están en el archivo **No Necesitan estructura auxiliar**

2. Cambiar el orden, u ordenarlo

Consigna es mostrarlo ordenado, si esta desordenado, o el el campo no se corresponde con el resultado
Se requiere una estructura auxiliar memoria para facilitar el procesamiento
Vector → conozca tamaño a priori
Cargo el archivo al vector → fread(v,sizeof(r), n,f);
Ordeno el vector → ordenarVector(v,n);
Muestro los datos del vector →for(i=0; i<n; i++) { cout<<v[i].c……);
Lista ordenada → No conozca tamaño a priori
Cargo el archivo a la lista ordenada →while(fread(&r, sizeof(r), 1,f)
insertarOrdenado(lista,r)
Muestro los datos de la lista→ while(lista!=NULL){r=pop(lista); cout<<r.c……..)

3. Ordenar un archivo desordenado

Dado un archivo de registros desordenado mostrar su contenido por pantalla ordenado por el campo1
Dado un archivo de registros desordenado ordenarlo por el campo1
Dado un archivo de registros desordenado mostrar su contenido por pantalla ordenado por el campo1 y a igualdad el campo 1 por el campo 2
v(j).c1>v(j+1).c1 || (v[j].c1==v[j+1].c1&&v[j].c2>v[j+1].c2)
Se requiere una estructura auxiliar memoria para facilitar el procesamiento

Dado un archivo desordenado ordenarlo
Vector → conozca tamaño a priori
Cargo el archivo al vector → visto
Ordeno el vector → visto
Me posiciono en la primera posición del archivo fseek(f,0,SEEK_SET)
Grabo los datos del vector → archivo se reemplaza y queda ordenado
fwrite(&r,sizeof(r),n,f)
Lista ordenada → No conozca tamaño a priori
Cargo el archivo a la lista ordenada
Me posiciono en la primera posición del archivo
Grabo los datos del vector → archivo se reemplaza y queda ordenado

---

Buscar en un archivo

1. Archivo ordenada tiene en la clave de búsqueda una PUP

No requiere estructura auxiliar → **búsqueda directa** Pos = Clave-valor 1ra pos
Si decido usarla puedo → vector No Lista→ solo permite secuencial
Clave 101 … 150 → buscar 135 → fseek(f, (135-101)*sizeof(r), SEEK_SET)

2. Archivo ordenado → la posicion no es predecible a priori

Una solución posible → **Busqueda binaria** en el archivo
Otra solución → cargarlo en estructura auxiliar y buscar en ella

3. Archivo desordenado

Necesariamente debemos cargarlo estructura auxiliar
Vector → si se da la condición del tamaño
Búsqueda secuencial
Ordenar el vector → búsqueda binaria → más eficiente que la secuencial
Cada posición solo necesitamos la información
Lista → si no se conoce el tamaño a priori
Búsqueda secuencial
Cada posición necesitamos la información + referencia al siguiente

Dados dos archivos uno con los datos personales de los alumnos, sin orden, el otro ordenado por legajo y materia con las materias cursadas Desarrollar un programa que muestre por pantalla los resultados de las materias y el nombre del estudiante. El campo que vincula ambos archivos es el numero de legajo

---

Datos están en dos estructuras y las quiero agrupar según un determinado orden

1. Dos archivos ordenados → intercalar conservando el orden

Apareo de los archivos sin necesidad de cargarlos en una estructura auxiliar

2. 1 archivo ordenado y el otro no

El ordenado lo dejo como esta
El desordenado lo cargo en una estructura auxiliar → vector ordenado o una lista ordenada
Una solución →apareo archivo ordenado con la estructura auxiliar
Otra solución → ordenar el archivo desordenado→ aparear los archivos

3. Ambos desordenados

Cargar ambos en una estructura auxiliar
Una solución → aparear las estructura auxiliares
Otra solución → ordenar los archivos → aparearlos
Otra solución → si alcanza → cargar ambos archivos misma estructura auxiliar y ordeno
Dados dos archivos ambos ordenados por numero de legajo, uno con los datos personales de los alumnos, el otro ordenado por legajo y materia con las materias cursadas Desarrollar un programa que muestre por pantalla los resultados de las materias y el nombre del estudiante ordenado por numero de legajo. El campo que vincula ambos archivos es el numero de legajo

Porque y cuando corte de control

1. Datos en un archivo ordenado por dos campos (legajo y materia) con repetición del legajo

Mostrar los datos agrupados por el primer campo → no necesito estructura auxiliar

Nl	mat	nota
4	Algo	1
4	Ssl	2
7	Algo	2
7	Md	10
7	syo	9

Mostrar los datos agrupados por nl → corte de control

NL 4
Mat nota
Algo 1
Ssl 2

Nl 7
Mat nota
Algo 2
Md 10
Syo 9

2. Archivo esta sin orden→ idéntica consigna

Cargarlo estructura auxiliar, ordenada por dos campos
Una solución→ hacer el corte de control en la estructura auxiliar
Otra solución → Ordenar el archivo, y corte de control en el archivo

Dni	Nl	cm	Nota

archivo

Dni	nl	mat	nota
6	14	Alg	2
9	25	Sint	4
3	18	Syo	3
2	30	md	1
5	35	alg	3
14	24	md	1

Son 6 registros
struct tr{int dni; int nl; char cm[20]; int nota};

tr vector[6];
tr r;
vector

Dni	nl	mat	nota

FILE* f = fopen(“nombre”,”rb+”);
I = 0;
1 fread(&r,sizeof(tr),1,f);
while(!feof(f)){
vector[i]= r;
i++;
fread(&r,sizeof(r),1,f);
}

2 while(fread(&r,sizeof(r),1,f)){
vector[i]= r;
i++;
}

3 for(i=0;i<6; i++){
fread(&r,sizeof(r),1,f);
vector[i]= r;
}

4 fread(&r,sizeof(r),1,f);
for(i=0;!feof(f); i++){
vector[i]= r;
fread(&r,sizeof(r),1,f);
}

5 fread(vector,sizeof(r),6,f);→ 5.1 fread(&vector[0],sizeof(r),6,f);
vector

Dni	nl	mat	nota
6	14	Alg	2
9	25	Sint	4
3	18	Syo	3
2	30	md	1
5	35	Alg	3
14	24	md	1

EN ESTE PUNTO VECTOR Y ARCHIVO DESORDENADO
ordenarVector(vector,6);// archivo sin orden, vector ordenado

Mostrar por pantalla sin modificar el archivo

```
for(i=0;i<6; i++)
cout <<vector[i].dni<< vector[i].nl……;
```

Reordenar el archivo

```
 fseek(f,0,SEEK_SET);
1 for(i=0;i<6; i++){
r = vector[i];
fwrite(&r,sizeof(r),1,f);
}

2 for(i=0;i<6; i++){
fwrite(&vector[i],sizeof(r),1,f);
}

3 fwrite(vector,sizeof(r),6,f);
```

Archivo ordenado

Volvemos al principio archivo sin orden y no se conoce el tamaño

Estructura adecuada lista

info				Sgte NODO*
dni	nl	mat	nota	

```
 struct Nodo{
          tr info;
          Nodo* sgte;
};
Nodo* Lista = NULL;

while(fread(&r,sizeof(r),1,f))
 insertarOrdenado(lista,r);
```

archivo sin orden. Pero la lista ordenada

mostrar fseek(f,o,SEEK_SET);

```
while(lista!=NULL){      ==
r = pop(lista);          ==
cout<<r.dni…….           fwrite(&r,sizeof(r),1,f)
}
```

A1

2
8
15

A2

3
4
5

R1

8

R2

5

A3

2
3
4
5
8
15

```
fread(&r1;sizeof(r1),1,A1);
fread(&r2;sizeof(r2),1,A2);
while(!feof(A1)&&!feof(A2)){
if(r1.nl<r2.nl){ fwrite(&r1;sizeof(r2),1,A3); fread(&r1;sizeof(r1),1,A1);}
 else fwrite(&r2;sizeof(r2),1,A3); fread(&r2;sizeof(r2),1,A2);}
}
// terminar el que no se agoto
while(!feof(A1)){
fwrite(&r1;sizeof(r2),1,A3);
fread(&r1;sizeof(r1),1,A1);
};
// terminar el que no se agoto
while(!feof(A2)){
fwrite(&r2;sizeof(r2),1,A3);
fread(&r2;sizeof(r1),1,A2);
}
```

Datos y dimensiones

Una dimensión

1- Conjunto definido a priori de Datos en memoria
   a. Vector estático **tipoRegistro Vector[N]**
      i. Forma de carga
         1. directa
         2. secuencial
         3. ordenada
         4. carga sin repetir
      ii. Origen del dato
         1. otra estructura
         2. dispositivo estándar de entrada cin>>
      iii. forma de ordenamiento
         1. directa PUP
         2. método de ordenamiento
      iv. búsquedas
         1. directa
         2. dicotomica

3. secuencial

v. recorridos
   1. secuencial
   2. directo
   3. con corte de control
   4. apareando

2- Conjunto definido a posteriori total al principio
   a. Vector dinamico **tipoRegistro* Vector = new tipoRegistro[N];**
      i. acciones idem anterior

3- Conjunto definido a posteriori con agregados individuales
   a. Lista simplemente enlazada **struct Nodo{Ti info, Nodo* sgte}; Nodo* Lista = NULL;**
      i. carga
         1. InsertarOrdenado
         2. InsertarSinRepetir
         3. push → caso particular de carga delante del primero
         4. queue → caso particular detrás del ultimo
      ii. origen del dato
         1. otra estructura
         2. dispositivo estándar de entrada
      iii. búsqueda
         1. secuencial
      iv. recorridos
         1. toda la estructura while(p)
         2. hasta apuntar al ultimo while(p->sgte)
         3. liberando las instancias while(lista){ x= pop(lista);……]
         4. con corte de control
         5. apareando

4- Definido a posteriori en el disco
   a. archivo binario **FILE* f = fopen(NombreFisico, modoApertura)**;
      i. carga - recorrido
         1. secuencial
         2. directa PUP
      ii. búsqueda
         1. directa
         2. binaria

Dos dimensiones

1- Ambas acotadas, numéricas y definidas en un subconjunto
   a. matriz dos dimensiones TD M[filas][columnas]

2- Ambas acotadas no definidas en un subconjunto
    a. matriz de dos dimensiones TD M[filas][columnas]
    b. Vector asociado td vf[filas]; tdv c[columnas]
        i. buscar en vector y asociar a matriz por posición
3- Una dimensión acotada la otra sin acotar
    a. vector de punteros Nodo* V[N]
        i. para la dimensión acotado ver lo anterior
        ii. para la dimensión no acotada una estructura enlazada
4- Ambas dimensiones sin acotar
    a. Lista de listas
        i. struct NodoLS{td info; NodoLS* sgte}
        ii. struct infoLP{td C1; NodoLS* proSL}
        iii. struct NodoLP{infoLP info; NodoLP* sgte}

Tres Dimensiones

1- Matriz de tres dimensiones
2- Matriz de dos dimensiones con un campo puntero a estructura enlazada
3- Vector de una dimensión con un campo puntero a una lista de listas
4- Lista de listas de listas

más de tres dimensiones

1- lo dejo a tu criterio → KOJ

Lista de listas

Estructuras de datos: Características

	Vector	Archivo	Listas ordenadas
Almacenamiento	Logico	fisico	Logico
Procesamiento	Rapido	Lento	Rapido
Tamaño en T.E.	Fijo	Variable	Variable
Tamaño cada posicion	Informacion	Informacion	Info + sgte
Busqueda directa	SI→V[N]	Fseek	Buscar secuencialmente
Persistencia	NO	SI	NO
Busqueda binaria	SI	SI	NO→arbol
Busqueda secuencial	SI	NO se recomienda	SI→unica posible
Carga directa	SI →V[N] = valor	Acceder → fseek Grabar → frwrte	Utilizando diferentes funciones
Carga secuencial	SI→v[ultimaPos] = valor	Acces. a ultima pos Grabar	SI Insertar ordenado
Carga sin repetir	Buscar si no esta se agrega	NO se recomienda Est. Auxiliares	Buscar si no esta se agrega
Ordenamiento	PUP Metodos de ordenam.	PUP Solo usando est. aux	insertarOrdenado
Definicion	Td Nombre[tamaño]	FILE*f; f= fopen(…..)	Definir Nodo → struct Punt Nodo* X = NULL
Acceso	V[N]	Acceder → fseek leerr → fread	Buscar secuencialmente
Corte de control	SI	SI	SI
Apareo	SI	SI	SI
Modificar la pos N	V[N] = valor	Apuntar → fseek Leer→fread Modificar en memoria Apuntar → fseek Grabar → fwrite	Buscar secuencialmente modificar

Estructuras de datos: Criterios de selección

4. Priorizar de ser posible acceso directo y velocidad de procesamiento.
    a. Vector en primer lugar si se cumple tamaño fijo, razonable y conocido a priori, y sin necesidad de persistencia→ acceso directo, búsqueda binaria, búsqueda secuencial
5. Si no se conoce el tamaño y no se requiere persistencia
    a. Estructuras enlazadas
        i. Pila si se debe invertir el orden o si es irrelevante
        ii. Colas si se debe mantener
        iii. Listas si se debe generar
6. Archivo si se requiere persistencia: directamente o estructuras auxiliares después cargar al archivo

Estructuras → muchos datos del mismo tipo

Archivos
Vectores → memoria → mejorar tiempos de procesamiento → tamaño fijo
Listas ordenadas → memoria →mejorar tiempos de procesamiento → tamaño variable

Observen como vienen los datos:

3. Es posible que los datos de partida estén en la forma en que nos piden los resultados→ procesamos directamente el archivo
4. Es posible que los datos de partida NO estén en la forma en que nos piden los resultados→ Debemos cargar en estructuras auxiliares y procesar esta estructura, reordenar los datos del archivo y procesar el archivo

Datos provienen de archivos

4. Tal como están se muestran

Consigna →Mostrar todos los datos del archivo
No requiere guardarlo en otra estructura, leer cada registro y mostrarlo
Tr v[10], r; v 10
FILE* f = fopen("MiArchivo.dat","rb+) while(fread(&r,sizeof(r),1,f)) { cout<<r.c…..}
Leer todos los registros y ponerlos en un vector fread(v,sizeof(r),10,f)
Dato Archivo ordenado
Consigna →Mostrar todos los datos del archivo ordenados por el mismo campo
No requiere guardarlo en otra estructura, leer cada registro y mostrarlo
Dado un archivo de registros mostrar su contenido por pantalla
Los resultados están en el orden que están en el archivo **No Necesitan estructura auxiliar**

5. Cambiar el orden, u ordenarlo

Consigna es mostrarlo ordenado, si esta desordenado, o el el campo no se corresponde con el resultado
Se requiere una estructura auxiliar memoria para facilitar el procesamiento
Vector → conozca tamaño a priori
Cargo el archivo al vector → fread(v,sizeof(r), n,f);
Ordeno el vector → ordenarVector(v,n);
Muestro los datos del vector →for(i=0; i<n; i++) { cout<<v[i].c……);
Lista ordenada → No conozca tamaño a priori
Cargo el archivo a la lista ordenada →while(fread(&r, sizeof(r), 1,f) insertarOrdenado(lista,r)
Muestro los datos de la lista→ while(lista!=NULL){r=pop(lista); cout<<r.c……..)

6. Ordenar un archivo desordenado

Dado un archivo de registros desordenado mostrar su contenido por pantalla ordenado por el campo1
Dado un archivo de registros desordenado ordenarlo por el campo1
Dado un archivo de registros desordenado mostrar su contenido por pantalla ordenado por el campo1 y a igualdad el campo 1 por el campo 2
v(j).c1>v(j+1).c1 || (v[j].c1==v[j+1].c1&&v[j].c2>v[j+1].c2)
Se requiere una estructura auxiliar memoria para facilitar el procesamiento

Dado un archivo desordenado ordenarlo
Vector → conozca tamaño a priori
Cargo el archivo al vector → visto
Ordeno el vector → visto
Me posiciono en la primera posición del archivo fseek(f,0,SEEK_SET)
Grabo los datos del vector → archivo se reemplaza y queda ordenado
fwrite(&r,sizeof(r),n,f)
Lista ordenada → No conozca tamaño a priori
Cargo el archivo a la lista ordenada
Me posiciono en la primera posición del archivo
Grabo los datos del vector → archivo se reemplaza y queda ordenado

---

Buscar en un archivo

4. Archivo ordenada tiene en la clave de búsqueda una PUP

No requiere estructura auxiliar → **búsqueda directa** Pos = Clave-valor 1ra pos
Si decido usarla puedo → vector No Lista→ solo permite secuencial
Clave 101 … 150 → buscar 135 → fseek(f, (135-101)*sizeof(r), SEEK_SET)

5. Archivo ordenado → la posicion no es predecible a priori

Una solución posible → **Busqueda binaria** en el archivo
Otra solución → cargarlo en estructura auxiliar y buscar en ella

6. Archivo desordenado

Necesariamente debemos cargarlo estructura auxiliar
Vector → si se da la condición del tamaño
Búsqueda secuencial
Ordenar el vector → búsqueda binaria → más eficiente que la secuencial
Cada posición solo necesitamos la información
Lista → si no se conoce el tamaño a priori
Búsqueda secuencial
Cada posición necesitamos la información + referencia al siguiente

Dados dos archivos uno con los datos personales de los alumnos, sin orden, el otro ordenado por legajo y materia con las materias cursadas Desarrollar un programa que muestre por pantalla los resultados de las materias y el nombre del estudiante. El campo que vincula ambos archivos es el numero de legajo

---

Datos están en dos estructuras y las quiero agrupar según un determinado orden

4. Dos archivos ordenados → intercalar conservando el orden

Apareo de los archivos sin necesidad de cargarlos en una estructura auxiliar

5. 1 archivo ordenado y el otro no

El ordenado lo dejo como esta
El desordenado lo cargo en una estructura auxiliar → vector ordenado o una lista ordenada
Una solución →apareo archivo ordenado con la estructura auxiliar
Otra solución → ordenar el archivo desordenado→ aparear los archivos

6. Ambos desordenados

Cargar ambos en una estructura auxiliar
Una solución → aparear las estructura auxiliares
Otra solución → ordenar los archivos → aparearlos
Otra solución → si alcanza → cargar ambos archivos misma estructura auxiliar y ordeno
Dados dos archivos ambos ordenados por numero de legajo, uno con los datos personales de los alumnos, el otro ordenado por legajo y materia con las materias cursadas Desarrollar un programa que muestre por pantalla los resultados de las materias y el nombre del estudiante ordenado por numero de legajo. El campo que vincula ambos archivos es el numero de legajo

Porque y cuando corte de control

3. Datos en un archivo ordenado por dos campos (legajo y materia) con repetición del legajo

Mostrar los datos agrupados por el primer campo → no necesito estructura auxiliar

nl	mat	nota
4	Algo	1
4	Ssl	2
7	Algo	2
7	Md	10
7	syo	9

Mostrar los datos agrupados por nl → corte de control

NL 4
Mat nota
Algo 1
Ssl 2

Nl 7
Mat nota
Algo 2
Md 10
Syo 9

4. Archivo esta sin orden→ idéntica consigna

Cargarlo estructura auxiliar, ordenada por dos campos
Una solución→ hacer el corte de control en la estructura auxiliar
Otra solución → Ordenar el archivo, y corte de control en el archivo

Dni	Nl	cm	Nota

archivo

dni	nl	mat	nota
6	14	Alg	2
9	25	Sint	4
3	18	Syo	3
2	30	md	1
5	35	alg	3
14	24	md	1

Son 6 registros

```
struct tr{int dni; int nl; char cm[20]; int nota};
tr vector[6];
tr r;
```

vector

dni	nl	mat	nota

```
FILE* f = fopen("nombre","rb+");
I = 0;
1 fread(&r,sizeof(tr),1,f);
while(!feof(f)){
 vector[i]= r;
i++;
 fread(&r,sizeof(r),1,f);
}

2 while(fread(&r,sizeof(r),1,f)){
 vector[i]= r;
i++;
}

3 for(i=0;i<6; i++){
fread(&r,sizeof(r),1,f);
vector[i]= r;
}

4 fread(&r,sizeof(r),1,f);
for(i=0;!feof(f); i++){
vector[i]= r;
fread(&r,sizeof(r),1,f);
}
```

5 fread(vector,sizeof(r),6,f);→ 5.1 fread(&vector[0],sizeof(r),6,f);

vector

dni	nl	mat	nota
6	14	Alg	2
9	25	Sint	4
3	18	Syo	3
2	30	md	1
5	35	Alg	3
14	24	md	1

EN ESTE PUNTO VECTOR Y ARCHIVO DESORDENADO

ordenarVector(vector,6);// archivo sin orden, vector ordenado
Mostrar por pantalla sin modificar el archivo
for(i=0;i<6; i++)
cout <<vector[i].dni<< vector[i].nl……;

Reordenar el archivo
fseek(f,0,SEEK_SET);
1 for(i=0;i<6; i++){
r = vector[i];
fwrite(&r,sizeof(r),1,f);
}

2 for(i=0;i<6; i++){
fwrite(&vector[i],sizeof(r),1,f);
}

3 fwrite(vector,sizeof(r),6,f);
Archivo ordenado

Volvemos al principio archivo sin orden y no se conoce el tamaño
Estructura adecuada lista

info				Sgte NODO*
dni	nl	mat	nota	

```
struct Nodo{
        tr info;
        Nodo* sgte;
};
Nodo* Lista = NULL;

while(fread(&r,sizeof(r),1,f))
 insertarOrdenado(lista,r);
```

archivo sin orden. Pero la lista ordenada

mostrar fseek(f,o,SEEK_SET);

```
while(lista!=NULL){      ==
r = pop(lista);          ==
cout<<r.dni…….           fwrite(&r,sizeof(r),1,f)
}
```

A1

2
8

A2

3
4

15

5

R1

8

R2

5

A3

2
3
4
5
8
15

```
fread(&r1;sizeof(r1),1,A1);
fread(&r2;sizeof(r2),1,A2);
while(!feof(A1)&&!feof(A2)){
if(r1.nl<r2.nl){ fwrite(&r1;sizeof(r2),1,A3); fread(&r1;sizeof(r1),1,A1);}
 else fwrite(&r2;sizeof(r2),1,A3); fread(&r2;sizeof(r2),1,A2);}
}
// terminar el que no se agoto
while(!feof(A1)){
fwrite(&r1;sizeof(r2),1,A3);
fread(&r1;sizeof(r1),1,A1);
};
// terminar el que no se agoto
while(!feof(A2)){
fwrite(&r2;sizeof(r2),1,A3);
fread(&r2;sizeof(r1),1,A2);
}
```

Datos y dimensiones

Una dimensión

5- Conjunto definido a priori de Datos en memoria
   a. Vector estático **tipoRegistro Vector[N]**
      i. Forma de carga
         1. directa
         2. secuencial
         3. ordenada
         4. carga sin repetir
      ii. Origen del dato
         1. otra estructura
         2. dispositivo estándar de entrada cin>>
      iii. forma de ordenamiento
         1. directa PUP
         2. método de ordenamiento

iv. búsquedas
    1. directa
    2. dicotomica
    3. secuencial
v. recorridos
    1. secuencial
    2. directo
    3. con corte de control
    4. apareando

6- Conjunto definido a posteriori total al principio
   a. Vector dinamico **tipoRegistro* Vector = new tipoRegistro[N];**
      i. acciones idem anterior
7- Conjunto definido a posteriori con agregados individuales
   a. Lista simplemente enlazada **struct Nodo{Ti info, Nodo* sgte}; Nodo* Lista = NULL;**
      i. carga
         1. InsertarOrdenado
         2. InsertarSinRepetir
         3. push → caso particular de carga delante del primero
         4. queue → caso particular detrás del ultimo
      ii. origen del dato
         1. otra estructura
         2. dispositivo estándar de entrada
      iii. búsqueda
         1. secuencial
      iv. recorridos
         1. toda la estructura while(p)
         2. hasta apuntar al ultimo while(p->sgte)
         3. liberando las instancias while(lista){ x= pop(lista);……]
         4. con corte de control
         5. apareando
8- Definido a posteriori en el disco
   a. archivo binario **FILE* f = fopen(NombreFisico, modoApertura)**;
      i. carga - recorrido
         1. secuencial
         2. directa PUP
      ii. búsqueda
         1. directa
         2. binaria

Dos dimensiones

5- Ambas acotadas, numéricas y definidas en un subconjunto
   a. matriz dos dimensiones TD M[filas][columnas]
6- Ambas acotadas no definidas en un subconjunto
   a. matriz de dos dimensiones TD M[filas][columnas]
   b. Vector asociado td vf[filas]; tdv c[columnas]
      i. buscar en vector y asociar a matriz por posición
7- Una dimensión acotada la otra sin acotar
   a. vector de punteros Nodo* V[N]
      i. para la dimensión acotado ver lo anterior
      ii. para la dimensión no acotada una estructura enlazada
8- Ambas dimensiones sin acotar
   a. Lista de listas
      i. struct NodoLS{td info; NodoLS* sgte}
      ii. struct infoLP{td C1; NodoLS* proSL}
      iii. struct NodoLP{infoLP info; NodoLP* sgte}

Tres Dimensiones

5- Matriz de tres dimensiones
6- Matriz de dos dimensiones con un campo puntero a estructura enlazada
7- Vector de una dimensión con un campo puntero a una lista de listas
8- Lista de listas de listas

más de tres dimensiones

2- lo dejo a tu criterio → KOJ

De vectores y matrices a Listas de listas

C1	C2	C3
0 1		
1 4		
2 3		
3 7		
4 5		

1) mostrar ordenados por el campo 1 cantidad de registros = valor conocido

guardar los datos en un vector
todos los datos
posición del archivo clave de busqueda

ordenar vector

mostrar los datos del vector
todos los datos → mostrarlos
busco en el vector → acceso directo al archivo a través de la posición

1	0
4	1
3	2
7	3
5	4

1	0
3	2
4	1
5	4
7	3

guardar los datos
i = 0;
while(fread(&r, sizeof(r), 1,f){
v[i] = r si guardo todos los datos
si solo quiero guardar la clave y el numero de registro
v[i].clave = r.c1;
v[i].pos = (ftell(f)/siseof(r)) -1;
i++;
}

ordenar vector
mostrar los datos si solo guarde la referencia al archivo
for (i=0, i<5;i++){
fseek(f,v[i].pos*sizeof(r), SEEK_SET);→ acceso directo pos
fread(&r, sizeof(r), 1, f); → leer registro
cout<<r.c1……
}

➔ si el tamaño se desconoce se guarda en una lista

C1	C2	nombre
1	5	Hola
4	3	Chau
1	8	huracan
7	6	Utn
4	1	Ceit

c1 1..10
c2 1..30
matriz

```
string matriz[10][30]
carga matriz
while(fread(&r, sizeof(r), 1,f){
matriz[r.c1][r.c2]=r.nombre;
}
mostrar
for(fila=0, fila < 10, fila++)
        for(col=0, col< 30, col++)
                cout<<m]i][j]
```

c1 1..10
c2 entero
vector de punteros

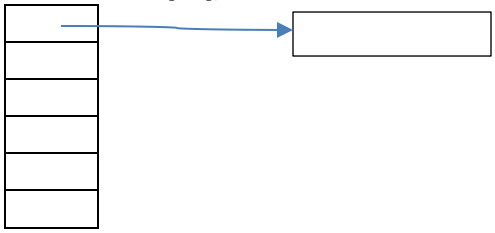
```
struct tipoinfo{
int c2;
string nombre;
}
```

C2 int	Nombre string

```
struct Nodo{
tipoinfo info;
Nodo* sgte;
}
```

Info		Sgte
C2	Nombre	

tipoinfo valor;
Nodo* vector[10];



```
for(i=0, i < 10, i++)
vector[i] = NULL;
cargar  vector de listas
while(fread(&r, sizeof(r), 1,f){
 valor.c2 = r.c2;
 valor.nombre = r.nombre;
 insertarordenado(vector[r.c1-1], valor);
}
for(i=0, i < 10, i++)0
 while(vector[i]!=NULL){
     valor = pop(vector[i]); cout<<valor.c2……;}
```

p; p->info;p->info.c1;p->info.prosl

c1 int, c2 int no acotados → lista de lista si el propósito es ordenar por dos campos

Info		Sgte nodo de la lista
C1 campo de orden	prosl al primer nodo de la LS	

Info		Sgte
C2 2do campo ord.	Nombre	

```
struct NodoLS{
tipoinfo info;
NodoLS* sgte;
};
tipoinfo valorLS;
struct TipoinfoLP{
int c1; TipoinfoLP info;

 NodoLS* prosl;
};

struct NodoLP{
NodoLP* sgte;
};
```

```
struct TipoInfoLP{
        int c1;
        NodoLS* prosl;
};
```

TipoinfoLP valorLP;
cargar los datos en una lista con sublista

```
while(fread(&r, sizeof(r), 1,f){
// cargar los registros de las listas
valorLS.c2 = r.c2; valorLS.nombre = r.nombre;
valorLP.c1 = r.c1; valorLP.prosl = NULL;
NodoLP * p=cargarsinrepetir(lista, valorLP);
insertarordenado(p->info.prosl  , valorLS);
};

while(lista!=NUll){
 valorLP = pop(lista);
        while(valorLP.prosl!=NULL){
        valorLS =  pop(valorLP.prosl):
        cout<< valorLS.c2…..
}
```

C1	prosl

Biblioteca de funciones de uso frecuente en la cátedra

En el presente documento se desarrollan las funciones de uso frecuente para la resolución de problemas de la cátedra de referencia. se abordan temas vinculados con estructuras de datos que permiten procesar conjunto de datos del mismo tipo: Vectores, Flujos, Estructuras enlazadas. La catedra utiliza C++ con el concepto de un C que toma de C++ solo alguna de las particularidades, no todas. En el caso de los punteros y el pasaje de parámetros por referencia se utiliza C++ por la facilidad que este lenguaje ofrece respecto de C. También se hace uso, aunque no se evalúa en los finales, de plantillas para la generalidad en el tipo de dato y punteros a funciones para la generalidad en el desarrollo de la función evitando quedar restringido a un único criterio de selección cuando corresponda.

El documento se estructura de la siguiente manera: primero se presentan las firmas de las funciones indicando valor de retorno, nombre y lista de tipos de datos de los parámetros. esto separado en grupos según el tipo de dato. Primero se abordan vectores, luego archivos y finalmente estructuras enlazadas, en dos columnas una para datos de tipo simple sin plantillas (en caso de uso requiere que se defina el tipo y se aclare el criterio en caso de corresponder), otra generalizando don plantillas y criterio cuando corresponda. Luego de la presentación de las firmas se desarrollan las funciones. también agrupadas por estructuras.

Consideraciones preliminares

En el tema de archivos trabajamos con archivos binarios con registros de tamaño fijo, con la modalidad de C (así está desarrollado este documento) y las cadenas utilizamos char*. Por eso usamos char* Nombre[tamaño] para cadenas y FILE*, fopen, fclose, fread, fwrite, fseek, ftell, también utilizamos las constantes SEEK_SET, SEEK_CUR, SEEK_END y el operador sizeof.

En cuanto a la declaración del nodo para estructuras enlazadas utilizamos

EL NODO	
Sin plantilla con struct en info	Con plantilla

struct tipoInfo{
  int c1;
  char * c2[20];
};
 	
struct Nodo{
  tipoInfo info; //int info con dato simple
  Nodo * sgte;
};
Nodo* Lista = NULL;
	template <typename T> struct Nodo{
	  T info;
	  Nodo<T>* sgte;
	};
	Nodo<tipoInfo>* Lista = NULL;

Firmas para vectores

agregar → Agrega un nodo al final del vector, el control de no superar el valor físico debe hacerse antes de invocar a la función, actualiza el tamaño lógico	
`void agregar(int arr[], int& n, int x)`	`template <typename T> void agregar(T arr[], int& n, T x)`
mostrar → muestra el contenido de un vector, en el caso de plantillas se agrega el concepto de puntero a función.	
`void mostrar(int v[], int n){`	`template <typename T> void mostrar(T v[], int n),void (*ver(T))){`
ordenar → ordena los elementos de un vector, creciente en enteros, según el criterio de ordenamiento y miembro con plantilla y puntero a función	
`void ordenar(int v[], int n,)`	`template <typename T> void ordenar(T v[], int n, int (*criterio)(T,T))`
buscar→ busca secuencialmente un elemento en un vector retornando el índice donde lo encuentra si el dato buscado esta, o el valor -1 silo buscado no está en el vector	
`//n tamaño lógico, x a buscar` `int buscar(int v[], int n, int x)`	`template <typename T, typename K>` `int buscar(T v[], int n, K x, int (*criterio)(T,K))`
busquedaBinaria→ busca dicotomicamente un elemento en un vector retornando el índice donde lo encuentra si el dato buscado esta, o el valor -1 silo buscado no está en el vector	
`int busquedaBinaria(int a[], int n, int v)`	`template<typename T, typename K>` `int busquedaBinaria(T a[], int n, K v, int (*criterio)(T, K))`

Firmas para archivos

leer → lee un registro de un archivo, retorna el registro leído	
`tipoReg leer(FILE* f)`	`template <typename T> T leer(FILE* f)`
leerN → lee N registros cargándolos en un vector	
`void leer**N**(FILE* f,tipoReg v, int N,)`	`template <typename T> void leer**N**(FILE* f, T v, int N)`
Grabar → graba un registro en un archive	
`void grabar(FILE* f, tipoReg r)`	`template <typename T> void grabar(FILE* f, T r)`
grabarN → graba N registros en un archivo desde un vector	
`void grabar(FILE* f, tipoReg v[], int N)`	`template <typename T> void grabar(FILE* f, T v[], int N)`
irA→ Accede en forma directa al registro indicado por un parámetro	
`void irA(FILE* arch, int n, int tamReg)`	`template <typename T> void irA(FILE* arch, int n){`

cantReg → indica la cantidad de registros de un archivo, volviendo el puntero a la posición en la que estaba en el momento de la invocación	
`long cantReg(FILE* f, int tamReg)`	`template <typename T> long cantReg(FILE* f)`
regActual → retorna el numero de registro en donde esta el puntero	
`long regActual(FILE*f, int tamReg))`	`template <typename T> long regActual(FILE* f){`
busquedaBinaria→	
`int busquedaBinaria(FILE* f, int x, int tamReg)`	`template <typename T, typename K> int busquedaBinaria(FILE* f, K x, int (*criterio)(T,K))`
ProcesarArch → procesa un archivo según e criterio de la función complementaria	
void procesarArch(FILE*f){	`template <typename T> void procesarArch(FILE*f,void (*procesar(T)))`

Firmas para estructuras enlazadas

push → inserta en una pila → delante del primer nodo	
`void push(Nodo*& p, int x)`	`template <typename T> void push(Nodo<T>*& p, T x)`
pop → elimina el primer nodo de la pila retornando el info que contenía.	
`int pop(Nodo*& p)`	`template <typename T> T pop(Nodo<T>*& p)`
Queue→ inserta un nodo en una cola → después del ultimo	
`void queue(Nodo& fte, Nodo*& fin, int x)`	`template <typename T> void queue(Nodo<T>*& fte, Nodo<T>*& fin, T x)`
unQueue → elimina el primer nodo de una cola retornando el info que contenia	
`int unQueue(Nodo*& fte, Nodo* & fin)`	`template <typename T> T unQueue(Nodo<T>& fte, Nodo<T>* & fin)`
insertaDelante → inserta en una lista ordenada el primer nodo o delante de este.	
`Nodo<T>*insertarDelante(Nodo*& p, int)`	`template <typename T> Nodo<T>*insertarDelante(Nodo<T>*& p, T x){`
insertaEnMedio → inserta en una lista ordenada en medio de dos valores. Retorna la dirección del nodo creado	
`Nodo* insertarEnMedio(Nodo*& p, int v)`	`template <typename T> Nodo<T>* insertarEnMedio(Nodo<T>*& p, T v, int (*critT,T)`
insertaAlFinal → inserta en una lista ordenada en la última posición. Retorna la dirección del nodo creado	
	`template <typename T>`

`Nodo* insertarAlFinal(Nodo*& p, int v,)`	`Nodo<T>* insertarAlFinal(Nodo<T>*& p, T v,)`
insertarOedenado → inserta en una lista según un criterio de ordenamiento	
`Nodo* insertarOrdenado(Nodo*& p, int v)`	`template <typename T> Nodo<T>* insertarOrdenado(Nodo<T>*& p, T v, int (*critT,T) )`
Buscar → busca en una lista un valor determinado. Retorna esta?direccion del Nodo: NULL	
`Nodo* buscar(Nodo* p, int v)`	`template <typename T, typename K> Nodo<T>* buscar(Nodo<T>* p, K v, int (*criterio)(T,K) )`
eliminarPrimero → elimina el primer nodo retornando su valor	
Aquí la funcion a utilizar es pop	Aquí la funcion a utilizar es pop
eliminarNodo→ Elimina un nodo con un valor determinado si lo encuentra retorna esta?1:0	
`int eliminarNodo(Nodo*& p, int v)`	`template <typename T, typename K> int eliminarNodo(Nodo<T>*& p, K v, int (*criterio)(T,K))`
InsertarSinRepetir → inserta solo si el valor no está en la lista R esta?la direcciónd onde lo encuentra: la dirección donde lo inserto	
`Nodo* insertarSinRepetir(Nodo*& p,int v)`	`template <typename T> Nodo<T>* insertarSinRepetir (Nodo<T>*& p,T v,int (*criterio)(T,T))`
CargaSinRepetir → carga un valor en una lista acumulando en caso que la clave exista	
Nodo* cargaSinRepetir(Nodo*& p, Nodo v, int x)	template <typename T, typename K> Nodo<T>* cargaSinRepetir(Nodo<T>*& p,T v,int (*criterio)(T,T),void (*agregar)(Nodo<T>*, K valor)){
eliminarNodos → elimina todos los nodos de una lista	
void eliminarNodos (Nodo* &p)	Template <typename T> void eliminarNodos (Nodo<T>* &p)
listaDeListas → carga en una lista ordenada por un campo con una sublista por otro	
Nodo* listaDeLista(Nodo* p, tipoLP vlp, tipos vls)	Template <typename T, typename K> Nodo<T>* listaDeListas(Nodo<T>*& p, T vlp, int(*critLP)(T,T),K vsl, int(*critLS)(K,K))

Plantillas para vectores

agregar → Agrega un nodo al final del vector, el control de no superar el valor físico debe hacerse antes de invocar a la función, actualiza el tamaño lógico

```
void agregar(int arr[], int& n,
int x){
   arr[n]=x;
   n++;

   return;
}


agregar(V, 10, 123)
```

```
template <typename T> void
agregar(T arr[], int& n, T v){
   arr[n]=v;
   n++;

   return;
}


agregar<tipoReg>(V, 10, reg)
agregar<int> (V, 10, 123)
```

mostrar → muestra el contenido de un vector, en el caso de plantillas de agrega el concepto de puntero a función.

```
void mostrar(int v[], int n){

   for(int i=0; i<n; i++){
      cout <<(v[i]<<endln;
   }

   return;
}
```

```
template <typename T> void
mostrar(T v[], int n),void
(*ver(T))){
   for(int i=0; i<n; i++){
      ver(v[i];
   }

   return;
}

void ver_reg(TipoReg r){
 cout << r.c1…….<<endl;
 return;
}
mostrar<tipoReg>(v, 10,ver_reg);
```

ordenar → ordena los elementos de un vector, creciente en enteros, según el criterio de ordenamiento y miembro con plantilla y puntero a funcion

```
void ordenar(int v[], int n,){

   int aux;
   int i,j,ord=0;
   for(i=0,i<n-1&&ord==0,i++){
      ord=1;
      for(int i=0; i<n-1; i++){

      if(v[j]>v[j+1]){
            aux = v[j];
            v[j] = v[j+1];
            v[j+1] = aux;
            ord = 1;
         }
      }
   }

   return;
}
```

```
template <typename T> void
ordenar(T v[], int n, int
(*criterio)(T,T)){
   T aux;
   int i,j,ord=0;
   for(i=0,i<n-1&&ord==0,i++){
      ord=1;
      for(int i=0; i<n-1; i++){

if(criterio(v[j],v[i+1])>0){
            aux = v[j];
            v[j] = v[j+1];
            v[j+1] = aux;
            ord = 1;
         }
      }
   }

   return;
}
int enteroCrec(int e1,inte2){
```

```
Ordenar(v,n);
```

```
    return e1>e2?1:0;
 }
ordenar<int>(v, n, enterocrec);
```

buscar→ busca secuencialmente un elemento en un vector retornando el índice donde lo encuentra, si el dato buscado esta o el valor -1 silo buscado no esta en el vector

```
int buscar(int v[], int n, int x)
{
   int i=0;
   while( i<n && (v[i]!= x)){
      i++;
   }
   return i<n?i:-1;
}

buscar(v,n, 123);
```

```
template <typename T, typename K>
int buscar(T v[], int n, K x, int
(*criterio)(T,K)){
   int i=0;
   while( i<n &&
criterio(v[i],x)>0 ){
      i++;
   }
   return i<n?i:-1;
}

int Legajo(tr r, int leg){
return r.leg!=leg?1:0;

buscar<tr,int>(v,n, 123, legajo);
```

busquedaBinaria→ busca dicotomicamente un elemento en un vector retornando el índice donde lo encuentra, si el dato buscado esta o el valor -1 silo buscado no esta en el vector

```
int busquedaBinaria(int a[], int
n, int v){
   int i=0;
   int j=n-1;
   int k;

  while( i<=j ){
      k=(i+j)/2;
      if( v > a[k] ){
         i=k+1;
      }
      else
      {
         if(v<a[k]){
            j=k-1;
         }
         else{
            return k;
         }
      }
   }
   return -1;
}
```

```
template<typename T, typename K>
int busquedaBinaria(T v[], int n,
K v, int (*criterio)(T, K)){
   int i=0;
   int j=n-1;
   int k;

  while( i<=j ){
      k=(i+j)/2;
      if( criterio(v,a[k])>0 ){
         i=k+1;
      }
      else
      {
         if(criterio(v,a[k])<0){
            j=k-1;
         }
         else{
            return k;
         }
      }
   }
   return -1;
}
```

Plantillas para archivos

leer → lee un registro de un archivo

```
tipoReg leer(FILE* f){
   tipoReg r;
   fread(&r,sizeof(r),1,f);
   return r;
}
```

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

leerN → lee N registros cargándolos en un vector

```
void leerN(FILE* f,tipoReg v, int
N,){
   fread(v,sizeof(v[0],N,f);
   return ;
}
```

```
template <typename T> void
leerN(FILE* f, T v, int N){

  fread(v,sizeof(T),N,f);
   return ;
}
```

Grabar → graba un registro en un archive

```
void grabar(FILE* f, tipoReg r){
   fwrite(&r,sizeof(r),1,f);
   return;
}
```

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

grabarN → graba N registros en un archivo desde un ventor

```
void grabar(FILE* f, tipoReg v[],
int N){
   fwrite(v,sizeof(tipoReg),N,f);
   return;
}
```

```
template <typename T> void
grabar(FILE* f, T v[], int N){

fwrite(v,sizeof(T),N,f);
   return;
}
```

irA→ Accede en forma directa al registro indicado por un parámetro

```
void irA(FILE* arch, int n, int
tamReg){
fseek(arch, n*tamReg,SEEK_SET);
 return;
}
```

```
template <typename T> void
irA(FILE* arch, int n){
 fseek(arch,
n*sizeof(T),SEEK_SET);
 return;
}
```

cantReg → indica la cantidad de registros de un archivo, volviendo el puntero a la posición en la que estaba en el momento de la invocación

```
long cantReg(FILE* f, int
tamReg){
   long cant, actual=ftell(f);
  fseek(f,0,SEEK_END);
  cant=ftell(f);
  fseek(f,actual,SEEK_SET);
```

```
template <typename T> long
cantReg(FILE* f){
   long cant, actual=ftell(f);
  fseek(f,0,SEEK_END);
  cant=ftell(f);
  fseek(f,actual,SEEK_SET);
```

```
   return cant /tamReg;
}
```

```
  return cant /sizeof(T);
}
```

regActual → retorna el numero de registro en donde esta el puntero

```
long regActual(FILE*f, int
tamReg)){
   return ftell(f)/tamReg;
}
```

```
template <typename T> long
regActual(FILE* f){

return ftell(f)/sizeof(T);
}
```

busquedaBinaria→

```
int busquedaBinaria(FILE* f, int
x){
   int i = 0;
  int j = cantReg(f)-1;
  tipoReg r;
   while(i<=j){
       k = (i+j)/2;
       irA(f,k, sizeof(r));
       r = leer(f);

     if( x>r.c){
         i = k+1;
       }
       else
       {if(x<r.c){
             j=k-1;
         } else return k;
       }
     }

   return -1;
}
```

```
template <typename T, typename K>
int busquedaBinaria(FILE* f, K x,
int (*criterio)(T,K)){
   int i = 0;
  int j = cantReg<T>(f)-1;
  T r;
   while( i<=j{
       k = (i+j)/2;
       irA<T>(f,k);
       r = leer<T>(f);

     if( criterio(x,r)>0 ){
         i = k+1;
       }
       else
       {if( criterio(x.r)<0 ){
             j=k-1;
         } else return k;
       }
     }

   return -1;
}
```

ProcesarArch → procesa un archivo según e criterio de la función complementaria

```
void procesarArch(FILE*f){
  while(r = leer(f)){
   cout<<r,c1…….<<endl;
 }

 return
```

```
template <typename T> void
procesarArch(FILE*f,void
(*procesar(T))){
   while(r = leer<T>(f)){
      procesar(r);
 }
 tipoReg r;


   return;
}
void ver_reg(TipoReg r){
 cout << r.c1…….<<endl;
 return;
}


procesarArch<TipoReg>(F, ver_reg)
```

Plantillas para estructuras enlazadas

El nodo

```
struct tipoInfo{
    int c1;
    char * c2[20];
};

struct Nodo{
    tipoInfo info; //int info con dato simple
    Nodo * sgte;
};
Nodo* Lista = NULL;
```

```
template <typename T> struct Nodo{
  T info;
  Nodo<T>* sgte;
};
Nodo<tipoInfo>* Lista = NULL;
```

push → inserta en una pila → delante del primer nono

```
void push(Nodo*& p, int x){
  Nodo* nuevo = new Nodo();
   nuevo->info = x;
   nuevo->sgte = p;
   p = nuevo;
   return;
}
```

```
template <typename T>
void push(Nodo<T>*& p, T x){
  Nodo<T>* nuevo = new Nodo<T>();
   nuevo->info = x;
   nuevo->sgte = p;
   p = nuevo;
   return;
}
```

pop → elimina el primer nodo de la pila retornando el info que contenía q

```
int pop(Nodo*& p){
   Nodo* q = p;
   int x = q->info;
   p = q->sgte;
   delete p;
   return x;
}
```

```
template <typename T>
T pop(Nodo<T>*& p){
   Nodo<T>* q = p;
   T x = q->info;
   p = q->sgte;
   delete p;
   return x;
}
```

Queue→ inserta un nodo en una cola → después del ultimo

```
void queue(Nodo& fte, Nodo*& fin,
int x){
  Nodo* nuevo = new Nodo();
   nuevo->info = x;
   nuevo->sgte = NULL;
   if(fte==NULL) fte = p;
   else fin->sgte = p;
   fin = p;
   return;
}
```

```
template <typename T>
void queue(Nodo<T>*& fte,
Nodo<T>*& fin, T x){
  Nodo<T>* nuevo = new Nodo<T>();
   nuevo->info = x;
   nuevo->sgte = NULL;
   if(fte==NULL) fte = p;
   else fin->sgte = p;
   fin = p;
   return;
}
```

unQueue → elimina el primer nodo de una cola retornando el info que contenia

```
int unQueue(Nodo*& fte, Nodo* &
fin){
   Nodo* q = fte;
   int x = q->info;
   fte = q->sgte;
```

```
template <typename T>
T unQueue(Nodo<T>*& fte, Nodo<T>*
& fin){
   Nodo<T>* q = fte;
   T x = q->info;
   fte = q->sgte;
```

```
   if(fte==NULL) fin = NULL;
   delete p;
   return x;
}
```

```
   if(fte==NULL) fin = NULL;
   delete p;
   return x;
}
```

insertaDelante → inserta en una lista ordenada el primer nodo o delante de este.

```
Nodo<T>*insertarDelante(Nodo*& p,
int x){
  Nodo* nuevo = new Nodo();
   nuevo->info = x;
   nuevo->sgte = p;
   p = nuevo;
return nuevo;}
```

```
template <typename T>
Nodo<T>*insertarDelante(Nodo<T>*&
p, T x){
  Nodo<T>* nuevo = new Nodo<T>();
   nuevo->info = x;
   nuevo->sgte = p;
   p = nuevo;
 return nuevo;}
```

insertaEnMedio → inserta en una lista ordenada en medio de dos valores. Retorna la dirección del nodo creado

```
Nodo* insertarEnMedio(Nodo*& p,
int v){
  Nodo* nuevo = new Nodo();
  nuevo->info = v;
  nuevo->sig = NULL;
  Nodo* aux = p;

   while( aux->sgte!=NULL && v <
aux->sgte.ifo){
      aux = aux->sig;
   }
 nuevo->sgte = aux->sgte;
 aux->sgte = nuevo;
return nuevo;}
```

```
template <typename T>
Nodo<T>*
insertarEnMedio(Nodo<T>*& p, T v,
int (*critT,T) ){
  Nodo<T>* nuevo = new Nodo<T>();
  nuevo->info = v;
  nuevo->sig = NULL;
  Nodo<T>* aux = p;
   while( aux->sgte!=NULL &&
crit(aux->sgte->info,v) > 0 ){
      aux = aux->sig;
   }
 nuevo->sgte = aux->sgte;
 aux->sgte = Nuevo;
   return nuevo;}
```

insertaAlFinal → inserta en una lista ordenada en la última posición. Retorna la dirección del nodo creado

```
Nodo* insertarAlFinal(Nodo*& p,
int v,){
  Nodo* nuevo = new Nodo();
  nuevo->info = v;
  nuevo->sgte = NULL;
if(p==NULL){
  p = nuevo; return nuevo;


else
  Nodo* aux = p;

   while( aux->sgte!=NULL){
      aux = aux->sgte;
   }
 aux->sgte = Nuevo;
   return Nuevo;
}
```

```
template <typename T>
Nodo<T>*
insertarAlFinal(Nodo<T>*& p, T
v,){
  Nodo<T>* nuevo = new Nodo<T>();
  nuevo->info = v;
  nuevo->sgte = NULL;
if(p==NULL){
  p = nuevo;
  return nuevo;
else
  Nodo<T>* aux = p;

   while( aux->sgte!=NULL){
      aux = aux->sgte;
   }
 aux->sgte = Nuevo;
   return nuevo;
}
```

insertarOedenado → inserta en una lista según un criterio de ordenamiento, en dato simple el criterio es creciente

```
// una opción simple
```

```
// una opción simple
```

```
Nodo* insertarOrdenado(Nodo*& p,
int v ){

if(p==NULL||v<p->info){
  p = insertardelante(p,v);
  else p = insertarEnMedio(p,v);
  return p;
}
// otra opción


Nodo* insertarOrdenado(Nodo*& p,
int v){
 Nodo* nuevo = new Nodo();
  nuevo->info = v;
  nuevo->sgte = NULL;
if(p==NULL||(v<p->info)){
  nuevo->sgte = p;p = nuevo;
  return nuevo;
else
  Nodo* aux = p;

   while(aux->sgte!=NULL)&&
v>aux->sgte.info){
      aux = aux->sig;
   }
 Nuevo->sgte = aux->sgte;
 aux->sgte = Nuevo;
   return nuevo;
}
```

```
template <typename T>
Nodo<T>*
insertarOrdenado(Nodo<T>*& p, T
v, int (*critT,T) ){
 if(p==NULL||crit(p->info,X)<=0){
  p = insertardelante(……)
  else p = insertarEnMedio(…..);
  return p;
}
// otra opción
template <typename T>
Nodo<T>*
insertarOrdenado(Nodo<T>*& p, T
v, int (*critT,T) ){
 Nodo<T>* nuevo = new Nodo<T>();
  nuevo->info = v;
  nuevo->sgte = NULL;
if(p==NULL||crit(p->info,v)<=0){
  nuevo->sgte =p;p = nuevo;
  return nuevo;
else
  Nodo<T>* aux = p;

   while(aux->sgte!=NULL)&&
crit(aux->sgte,v)>0){
      aux = aux->sig;
   }
 Nuevo->sgte = aux->sgte;
 aux->sgte = Nuevo;
   return nuevo;
}
```

Buscar → busca en una lista un valor determinado. Retorna esta?direccion del Nodo: NULL

```
Nodo* buscar(Nodo* p, int v)

{
   Nodo* aux = p;
   while( aux!=NULL && aux->info
!=v ){
     aux = aux-0>sig;

 }
   return aux;
}
```

```
template <typename T, typename K>
Nodo<T>* buscar(Nodo<T>* p, K v,
int (*criterio)(T,K) )
{
   Nodo<T>* aux = p;
   while( aux!=NULL &&
criterio(aux->info,v)!=0 ){
      aux = aux->sig;
   }

   return aux;
}
```

eliminarPrimero → elimina el primer nodo retornando su valor

Aquí la funcion a utilizar es pop	Aquí la funcion a utilizar es pop

eliminarNodo→ Elimina un nodo con un valor determinado si lo encuentra retorna esta?1:0

```
int eliminarNodo(Nodo*& p, int v)
{
   Nodo* aux = p;
   Nodo* ant = NULL;
   while(aux!=NULL&&aux->info!=v{
      ant = aux;
      aux = aux->sgte;
```

```
template <typename T, typename K>
int eliminarNodo(Nodo<T>*& p, K
v, int (*criterio)(T,K))
{
   Nodo<T>* aux = p;
   Nodo<T>* ant = NULL;
   while( aux!=NULL &&
criterio(aux->info,v)!=0 ){
```

```
    }

    If(aux==NULL) return 0;
    if( ant!=NULL ){
       ant->sig = aux->sig;
    }
    else {
       p = aux->sig;
    }
    delete aux;
   return 1;
}
```

```
       ant = aux;
       aux = aux->sgte;
    }
    If(aux==NULL) return 0;
    if( ant!=NULL ){
       ant->sig = aux->sig;
    }
    else {
       p = aux->sig;
    }
    delete aux;
   return 1;
}
```

InsertarSinRepetir → inserta solo si el valor no está en la lista R esta?la direcciónde lo encuentra: la dirección donde lo inserto

```
Nodo* insertarSinRepetir(Nodo*& p,
int v)){


Nodo* q = buscar(p,v);
   If(x==NULL)
     q = insertarOrdenado(p,v);


return q;
}
```

```
template <typename T>
Nodo<T>*
insertarSinRepetir(Nodo<T>*& p,T
v,int (*criterio)(T,T)){
   Nodo<T>* q =
buscar<T,T>(p,v,criterio);
   If(x==NULL)
     q = insertarOrdenado<T>
(p,v,criterio);

   return q;
}
```

CargaSinRepetir → carga un valor en una lista acumulando en caso que la clave exista

```
Nodo* cargaSinRepetir(Nodo*& p, Nodo v,
int x) ){

Nodo* p = insertarSinRepetir(p, v);
p->info.campo+=x;
return p;
}
```

```
template <typename T, typename K>
Nodo<T>* cargaSinRepetir(Nodo<T>*& p,T
v,int (*criterio)(T,T),void
(*agregar)(Nodo<T>*, K valor) ){
Nodo<T>* p = insertarSinRepetir(…….);
Agregar(p,valor);
return p;
}
```

eliminarNodos → elimina todos los nodos de una lista

```
void eliminarNodos (Nodo* &p){
  Nodo* q;
  while(p){
    q=p;
    p=p->sgte;
    delete(q);
  }
    return;
 }
```

```
Template <typename T> void eliminarNodos
(Nodo<T>* &p){
  Nodo<T>* q;
  while(p){
    q=p;
    p=p->sgte;
    delete(q);
  }
    return;
 }
```