Algoritmos y estructura de datos

Asignatura anual, código 082021

MACHETE MODULO 3 Struct y FILE

Departamento de Ingeniería en Sistemas de Información
Universidad Tecnológica Nacional FRBA

Tabla de contenido

Estructura de datos: Flujos y Registros

1

Estructura tipo Registro

```
struct NombreTipo {
        Tipo Identificador;
        Tipo Identificador;
}
struct TipoRegistro {
        int N;
        double Y;
};                      //      declara un tipo
TipoRegistro Registro; //       define una variable
```

Ejemplo de estructuras anidadas en C

```
struct TipoFecha {
        int     D;
        int     M;
        int     A;
};                      //      declara un tipo fecha

struct TipoAlumno {
        int             Legajo;
        string          Nombre;
        TipoFecha       Fecha
};                      //      declara un tipo Alumno con un campo de tipo Fecha
TipoAlumno Alumno;
```

Estructura tipo Archivo[1]

FILE * F; asocia al identificador F que contiene información para procesar un 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

[1] En Algoritmos este año trabajaremos con estructuras FILE *

Operaciones simples

Función	Descripción
FILE *f;	Define f como puntero a FILE
f = fopen (“archivo”, “wb”);	Asocia f a un flujo
f = fropen(“archivo”, “wb”);	Similar anterior, si esta abierto antes lo cierra
fclose(f);	Vacía el flujo y cierra el archivo asociado
fflush(f);	Produce el vaciado de los flujos
remove(“archivo”);	El archivo ya no queda accesible
rename(“viejo”, “nuevo”);	Renombra con nuevo el viejo nombre
sizeof(tipo)	Retorna el tamaño de un tipo o identificador
SEEK_CUR	Constante asociada a fseek (lugar actual)
SEEK_END	Constante asociada a fseek (desde el final)
SEEK_SET	Constante asociada a fseek (desde el inicio)
size_t fread(&r, tam,cant, f)	Lee cant bloques de tamaño tam del flujo f
size_t fwrite(&r,tam,cant,f)	Graba cant bloques de tamaño tam del flujo f
fgetpos(f, pos)	Almacena el valor actual del indicador de posicion
fsetpos(f,pos)	Define el indicador de posicion del archive en pos
ftell(f)	El valor actual del indicador de posición del archivo
fseek(f, cant, desde)	Define indicador de posicion a partir de una posicion.

Template: read

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

Template: write

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

Template: seek

```
template <typename T> void seek(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);
}
```

Template: fileSize

```
template <typename T> long fileSize(FILE* f)
{
   // tomo la posicion actual
```

```
      long curr=ftell(f);
    // muevo el puntero al final del archivo
      fseek(f,0,SEEK_END);  // SEEK_END hace referencia al final del archivo
  // tomo la posicion actual (ubicado al final)
  long ultimo=ftell(f);
// vuelvo a donde estaba al principio
  fseek(f,curr,SEEK_SET);

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

Template: filePos

```
template <typename T> long filePos(FILE* arch)
{
   return ftell(arch)/sizeof(T);
}
```

Template: busquedaBinaria

El algoritmo de la búsqueda binaria puede aplicarse perfectamente para emprender búsquedas sobre los registros de un archivo siempre y cuando estos se encuentren ordenados.

```
template <typename T, typename K>
int busquedaBinaria(FILE* f, K v, int (*criterio)(T,K))
{  // indice que apunta al primer registro
   int i = 0;
   // indice que apunta al ultimo registro
   int j = fileSize<T>(f)-1;
   // calculo el indice promedio y posiciono el indicador de posicion
   int k = (i+j)/2;
   seek<T>(f,k);
   // leo el registro que se ubica en el medio, entre i y j
   T r = leerArchivo<T>(f);

   while( i<=j && criterio(r,v)!=0 )
   {
     // si lo que encontre es mayor que lo que busco...
      if( criterio(r,v)>0 )
      {
         j = k-1;
      }
      else
      {
       // si lo que encontre es menor que lo que busco...
         if( criterio(r,v)<0 )
         {
            i=k+1;
         }
      }
      // vuelvo a calcular el indice promedio entre i y j
      k = (i+j)/2;
     // posiciono y leo el registro indicado por k
      seek<T>(f,k);
// leo el registro que se ubica en la posicion k
      r = leerArchivo<T>(f);
   }
// si no se cruzaron los indices => encontre lo que busco en la posicion k
   return i<=j?k:-1;
}
```

Ejemplos

Leer un archivo de registros usando el *template* `read`.

```
f = fopen("PERSONAS.DAT","r+b");
// leo el primer registro
Persona p = read<Persona>(f);
while( !feof(f) )
{
   cout << p.dni<<", "<<p.nombre<<", "<<p.altura << endl;
   p = read<Persona>(f);
}

fclose(f);
```

Escribir registros en un archivo usando el *template* `write`.

```
f = fopen("PERSONAS.DAT","w+b");
// armo el registro
Persona p;
p.dni = 10;
strcpy(p.nombre,"Juan");
p.altura = 1.70;
// escribo el registro
write<Persona>(f,p);
fclose(f);
```

Acceso directo a los registros de un archivo usando los templates `fileSize`, `seek` y `read`.

```
f = fopen("PERSONAS.DAT","r+b");
// cantidad de registros del archivo
long cant = fileSize<Persona>(f);

for(int i=cant-1; i>=0; i--)
{
  // acceso directo al i-esimo registro del archivo
   seek<Persona>(f,i);

   Persona p = read<Persona>(f);

   cout << p.dni<<", "<<r.nombre<<", "<< r.altura << endl;
}

fclose(f);
```

Introducción a Streams en C++

2

Lectura y Escritura en Archivos de Bloques a través de Flujos

El lenguaje C++ y su biblioteca estándar proveen abstracciones para el manejo de flujos (streams) que se conectan a archivos (files). El encabezado que declara estas abstracciones es <fstream>, de file-stream.

Esta sección presenta como

- Crear un flujo desde o hacia un archivo.
- Leer y escribir a través de un flujo
- Manejar los indicadores de posición de lectura y escritura de un flujo
- Cerrar un flujo

Archivos de Bloques de Tamaño Constante

Una de las formas de almacenar datos en archivos es mediante una secuencia de bloques, donde todos los bloques tienen la misma cantidad de bytes. Un bloque de bytes puede almacenar cualquier valor que necesitemos, pero en esta sección asumiremos que un bloque tiene un solo registro representado por un struct.

Por ejemplo, para almacenar en un archivo las temperaturas registradas en distintos puntos de una superficie, podemos diseñar un bloque que contenga la abscisa, la ordenada, y la temperatura registrada. Así, el archivo contendría una secuencia bloques, todos con tres datos reales, y, por lo tanto, del mismo tamaño.

Los bloques los declaramos como structs, y para conocer el tamaño en bytes de un struct aplicamos el operador sizeof.

Para presentar el procesamiento de un archivo de bloques de tamaño constantes, vamos a ver un programa simple que mediante flujos escribe datos, lee datos, y reestablece el indicador de posición de lectura para manejar los datos de curso universitarios.

La directiva

```
#include <fstream>
```

incluye las declaraciones necesarias para manejar las abstracciones de flujo de archivos.

Luego declaramos una estructura que establece la forma de los bloques que vamos a leer y escribir, y creamos una variable en base a esa estructura.

```
struct Curso {
      char especialidad;
      int codigo;
      int nivel;
      int alumnos;
      double promedio;
} curso;
```

El programa crea un nuevo archivo cursos, donde escribiremos cuatro cursos ejemplos. Para eso, necesitamos declarar una variable del tipo ofstream (output-file-stream) que nos va a permitir escribir bloques.

```
ofstream out("cursos", ios::binary);
```

El nombre de la variable es out, el nombre del archivo es cursos, y el modo binary asegura que la cantidad de bytes escritos sea constante. La variable out queda inicializada es el flujo que conecta el programa al archivo cursos.

A continuación, asignamos a los miembros de curso valores ejemplos.

```
curso.especialidad = 'K',
curso.codigo = 1051,
curso.nivel = 1;
curso.alumnos = 29;
curso.promedio = 7.8;
```

Para escribir en el flujo out el bloque curso invocamos a la función writeblock

```
writeblock(out, curso);
```

Luego escribimos otros tres cursos ejemplo. Por último, cerramos el flujo mediante

```
out.close();
```

Ahora volvemos a conectarnos al archivo, esta vez para lectura.

```
ifstream in("cursos", ios::binary);
```

La variable in es del tipo ifstream (intput-file-stream), el cual nos permite leer bloques.

La lectura la realizamos mediante un ciclo que mientras haya bloques en in, lea un bloque y envíe su contenido por cout; en este ejemplo, solo muestra los cursos con más de 25 alumnos. El psueudocódigo es el siguiente:

```
while( haya otro curso en in) )
      if( curso.alumnos > 25 )
            mostrar el curso por cout
```

La expresión

```
readblock(in, curso)
```

lee un bloque desde in y lo almacena en curso, el valor de retorno es in, el cual puede ser utilizado como un valor boolean en la expresión de control de while. El ciclo completo es el siguiente:

```
while( readblock(in, curso) )//al llegar al fin la llamada devuelve el apuntador nulo
          if( curso.alumnos > 25 )
                cout
                      << "Especialidad: " << curso.especialidad << ", "
                      << "      Código: " << curso.codigo       << ", "
                      << "       Nivel: " << curso.nivel        << ", "
                      << "     Alumnos: " << curso.alumnos      << ", "
                      << "    Promedio: " << curso.promedio     << endl;
```

Por último, cerramos el flujo con la sentencia

```
    in.close();
```

Las funciones template writeblock y readblock abstraen y facilitan la lectura de bloques.

Este es el programa completo, incluyendo la definición de writeblock y readblock.

```
/* Escribe y lee archivo de registros
 * Genera un archivo con registros de cursos, luego abre ese archivo y
muestra los cursos de segundo año con cantidad de alumnos mayor a 25.
 * 20130424
 * JMS
 */

#include <iostream>
#include <fstream>

template<typename T>
std::ostream& writeblock(std::ostream& out, const T& block){
    return out.write(
                reinterpret_cast<const char*>(&block),
                sizeof block
          );
}

Conversión entre tipos con reinterpret_cast
Escritura de bytes mediante la función miembro write de ostream: Envia un
número fijo de bytes empezando en una ubicación especifica de memoria al
flujo especificado. La función read de istream recibe un número fijo de
bytes del flujo especificado y los coloca en un area de memoria que
empieza en una dirección especificada. Al escribir el entero numero en un
archivo en lugar de escribir archivoSalida << numero; se puede escribir la
versión binaria
archivoSalida.write(reinterpret_cast<const char *>(&numero), sizeof (numero)

template<typename T>
std::istream& readblock(std::istream& in, T& block){
    return in.read(
                reinterpret_cast<char*>(&block),
                sizeof block
          );
}
```

```
int main(){
    using namespace std;

    struct Curso {
          char especialidad;
          int codigo;
          int nivel;
          int alumnos;
          double promedio;
    } curso;

    ofstream out("cursos", ios::binary);

    curso.especialidad = 'K',
    curso.codigo = 1051,
    curso.nivel = 1;
    curso.alumnos = 29;
    curso.promedio = 7.8;
    writeblock(out, curso);

    curso.especialidad = 'R',
    curso.codigo = 4152,
    curso.nivel = 4;
    curso.alumnos = 41;
    curso.promedio = 6.9;
    writeblock(out, curso);

    curso.especialidad = 'K',
    curso.codigo = 2051,
    curso.nivel = 1;
    curso.alumnos = 22;
    curso.promedio = 6.7;
    writeblock(out, curso);

    curso.especialidad = 'K',
    curso.codigo = 2011,
    curso.nivel = 1;
    curso.alumnos = 26;
    curso.promedio = 7.9;
    writeblock(out, curso);

    out.close();

    ifstream in("cursos", ios::binary);


    while( readblock(in, curso) )
          if( curso.alumnos > 25 )
                cout
                      << "Especialidad: " << curso.especialidad << ", "
                      << "      Código: " << curso.codigo       << ", "
                      << "       Nivel: " << curso.nivel        << ", "
                      << "     Alumnos: " << curso.alumnos      << ", "
                      << "    Promedio: " << curso.promedio     << endl;
```

```
    in.close();
}

Salida:
Especialidad: K,       Codigo: 1051,        Nivel: 1,      Alumnos: 29,
Promedio: 7.8
Especialidad: R,       Codigo: 4152,        Nivel: 4,      Alumnos: 41,
Promedio: 6.9
Especialidad: K,       Codigo: 2011,        Nivel: 1,      Alumnos: 26,
Promedio: 7.9
```

Funciones miembro para posicionar el puntero
seekg (obtener desde donde se hara la próxima entrada, para istream), seekp (colocar, el byte donde se colocara la próxima salida para ostream), tellg, tellp (para determinar las actuales posiciones del puntero obtener y colocar). Se puede indicar un segundo argumento con la dirección de búsqueda que puede ser ios::beg (opción predeterminada desde el inicio) ios::cur para un posicionamiento relativo a la posición actual. ios::end para un posicionamiento relativo al final.
archivo.seekg(n); // desplaza n bytes desde el inicio
archivo.seekg(n, ios::beg); // idem anterior
archivo.seekg(n, ios::end); //se posiciona n bytes hacia atras desde el fin
archivo.seekg(n, ios::cur); se desplaza n bytes desde la posición actual

C++ ifstream, ofstream y fstream

C++ para el acceso a ficheros de texto ofrece las clases ifstream, ofstream y fstream. (**i** input, **f** file y **stream**). (**o** output). fstream es para i/o.

Abrir los ficheros

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
/* pasando parámetros en la declaración de la variable */
   ifstream f("fichero.txt", ifstream::in);

/* se abre después de declararlo, llamando a open() */
   ofstream f2;
   f2.open("fichero2.txt", ofstream::out);
```

El primer parámetro es el nombre del fichero, el segundo el modo de apertura.

- **app** (**app**end) Para añadir al final del fichero. Todas las escrituras se hacen al final independiente mente de la posición del puntero.
- **ate** (**at e**nd). Para añadir al final del fichero. En caso de mover el puntero, la escritura se hace donde esta el mismo.
- **binary** (**binary**) Se abre el fichero como fichero binario. Por defecto se abre como fichero de texto.
- **in** (**in**put) El fichero se abre para lectura.

- **out** (**out**put) El fichero se abre para escritura
- **trunc** (**trunc**ate) Si el fichero existe, se ignora su contenido y se empieza como si estuviera vacio. Posiblemente perdamos el contenido anterior si escribimos en él.

Se puede abrir con varias opciones con el operador OR o el caracter | .

```
f2.open("fichero2.txt", ofstream::out | ofstream::trunc);
```

Hay varias formas de comprobar si ha habido o no un error en la apertura del fichero. La más cómoda es usar el operator ! que tienen definidas estas clases. Sería de esta manera

```
if (f)
{
   cout << "fallo" << endl;
   return -1;
}
```

!f (no f) retorna true si ha habido algún problema de apertura del fichero.

Leer y escribir en el fichero

Existen metodos específicos para leer y escribir bytes o texto: get(), getline(), read(), put(), write(). Tambien los operadores **<<** y **>>**.

```
/* Declaramos un cadena para leer las líneas */
char cadena[100];
...
/* Leemos */
f >> cadena;
...
/* y escribimos */
f2 << cadena;
/*Copiar un archivo en otro */
/* Hacemos una primera lectura */
f >> cadena;/*Lectura anticipada controla si es fin de archivo*/
while (!f.feof()){
   /* Escribimos el resultado */
   f2 << cadena << endl;
   /* Leemos la siguiente línea */
   f >> cadena;
}
```

Cerrar los ficheros

```
f.close(); f2.close();
```

Ejemplos Archivos de texto

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
   char cadena[128];
   // Crea un fichero de salida
   ofstream fs("nombre.txt");
   // Enviamos una cadena al fichero de salida:
   fs << "Hola, mundo" << endl;
   // Cerrar el fichero para luego poder abrirlo para lectura:
   fs.close();

   // Abre un fichero de entrada
```

```
   ifstream fe("nombre.txt");
   // Lectura mediante getline
   fe.getline(cadena, 128);
   // mostrar contenido por pantalla
   cout << cadena << endl;

   return 0;
}

int main() {
   char cadena[128];
   ifstream fe("streams.cpp");
   while(!fe.eof()) {
      fe >> cadena;
      cout << cadena << endl;
   }
   fe.close();
   return 0;}
```

Ejemplo Archivo binario

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
struct tipoReg {
   char nombre[32];
   int edad;
   float altura;
};
int main() {
   tipoReg r1;
   tipoReg r2;
   ofstream fsalida("prueba.dat",ios::out | ios::binary);
   strcpy(r1.nombre, "Juan");
   r1.edad = 32;
   r1..altura = 1.78;

   fsalida.write(reinterpret_cast<char *>(&r1),    sizeof (tipoReg));

fsalida.close();// lo cerramos para abrirlo para lectura
   ifstream fentrada("prueba.dat", ios::in | ios::binary);
   fentrada.read(reinterpret_cast<char *>(&r2), sizeof(tipoReg));
   cout << r2.nombre << endl;
   cout << r2.edad << endl;
   cout << r2.altura << endl;
   fentrada.close();
return 0;
}
```

Acceso directo

```
#include <fstream>
using namespace std;
int main() {
   int i;
   char mes[][20] = {"Enero", "Febrero", "Marzo","Abril", "Mayo",
"Junio", "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre",
      "Diciembre"};
   char cad[20];

   ofstream fsalida("meses.dat",ios::out | ios::binary);

   // Crear fichero con los nombres de los meses:
   cout << "Crear archivo de nombres de meses:" << endl;
   for(i = 0; i < 12; i++)
      fsalida.write(mes[i], 20);
   fsalida.close();
   ifstream fentrada("meses.dat", ios::in | ios::binary);
   // Acceso secuencial:
   cout << "\nAcceso secuencial:" << endl;
   fentrada.read(cad, 20);
   do {
      cout << cad << endl;
      fentrada.read(cad, 20);
   } while(!fentrada.eof());

   fentrada.clear();
   // Acceso aleatorio:
   cout << "\nAcceso aleatorio:" << endl;
   for(i = 11; i >= 0; i--) {
      fentrada.seekg(20*i, ios::beg);
      fentrada.read(cad, 20);
      cout << cad << endl;
   }

   // Calcular el número de elementos
   // almacenados en un fichero:
   // ir al final del fichero
   fentrada.seekg(0, ios::end);
   // leer la posición actual
   pos = fentrada.tellg();
   // El número de registros es el tamaño en
   // bytes dividido entre el tamaño del registro:
   cout << "\nNúmero de registros: " << pos/20 << endl;
   fentrada.close();

   return 0;
}
```

Funciones miembros de la clase stream

Funcion	Descripcion
bad	true si ha ocurrido un error
clear	limpia las banderas de estado (status flags)
close	cierra un stream
eof	true si se alcanzó el fin de archivo
fail	true si ha ocurrido un error
fill	establecer manipulador de carácter de relleno
flags	accesa o manipula las banderas de formato de un stream
flush	vaciar el buffer de un stream
gcount	número de caracteres leidos durante la última operación de entrada
get	lectura de caracteres
getline	lectura de una línea de caracteres
good	true si no ha ocurrido un error
ignore	leer y descartar caracteres
open	abrir un stream de entrada y/o salida
peek	verifica la siguiente entrada de carácter
precision	manipula la precisión del stream
put	escritura de caracteres
putback	regresar caracteres al stream
rdstate	regresa la bandera de estado de stream
read	lee datos de un stream hacia un buffer
seekg	realiza acceso aleatorio sobre un stream de entrada
seekp	realiza acceso aleatorio sobre un stream de salida
setf	cambiar las banderas de formato
tellg	lee el puntero del stream de entrada
tellp	lee el puntero del stream de salida
unsetf	limpiar las banderas de formato
width	accesa y manipula la longitud minima del campo
write	escritura datos desde un buffer hacia un stream

Consideraciones sobre stream.

El ciclo

```
while (!arch1.eof () && !arch2.eof ()) {… };
```

En forma mas compacta se puede escribir:

```
while(arch1 and  arch2){…}
```

eof no es la única condición de fin, además, la notación anterior induce a que se asuma que hay una marca eof. Por otro lado, C++ permite usar streams como booleans, el stream es verdadero si se puede seguir leyendo o falso en caso contrario. Lo que nos conduce a la segunda forma de escritura más compacta y comprensible.

Ejemplo de corte de control

Ejemplo de control break sobre un campo de control.
Entrada 3-uplas (pedido, cliente, importe), ordenadas por cliente. Ejemplo:
20 78 10.5; 51 78 9.5; 12 78 5.5; 26 80 1.5; 63 80 20.5
Salida 2-uplas (cliente, total), ordenandas por cliente; y total general. Ejemplo:
78 25.5; 80 22

```
// programa que opera sobre stdin
#include <iostream>
int main() {
struct { unsigned id, cliente; double importe;} pedido;

double totalGeneral=0;
std::cin >> pedido.id >> pedido.cliente >> pedido.importe;
while (std::cin) {
  unsigned previo = pedido.cliente;
  double total = pedido.importe;
   while (std::cin >> pedido.id >> pedido.cliente >> pedido.importe and pedido.cliente==previo) {
     total+=pedido.importe;
   }
  std::cout << previo << '\t' << total <<'\n';
  totalGeneral+=total;
}
std::cout << totalGeneral <<'\n';
return 0;
}
```

```
// programa que opera sobre un flujo binario
#include <iostream>
#include <fstream>
int main() {
struct { unsigned id, cliente; double importe;} pedido;
ifstream flectura("prueba.dat",ios::in | ios::binary);
double totalGeneral=0;
readblock(flectura, pedido);
while (not flectura) {
  unsigned previo = pedido.cliente;
  double total = pedido.importe;
   while (readblock(flectura, pedido) and pedido.cliente==previo) {
     total+=pedido.importe;
   }
   std::cout << previo << '\t' << total <<'\n';
   totalGeneral+=total;
 }
std::cout << totalGeneral <<'\n';
flectura.close();
return 0;
}
```

Ejemplo de Apareo

Ejemplo de apareo sobre un campo, sin repetición, por el que esta ordenado.
Entrada 3-uplas (pedido, cliente, importe), ordenadas por cliente.

```
// programa que opera sobre flujos binarios
#include <iostream>
#include <fstream>
int main() {
struct { unsigned id, cliente; double importe;} pedido1,pedido2;
ifstream flectura1("prueba1.dat",ios::in | ios::binary);
ifstream flectura2("prueba2.dat",ios::in | ios::binary);
ofstream fmezcla("prueba.dat",ios::out | ios::binary);
readblock(flectura1, pedido1);
readblock(flectura2, pedido2);
while (flectura1 or flectura2) {
   if(!flectura2 or (flectura1 and pedido1.cliente < pedido2.cliente))
   //si el segundo no tiene  o teniendo ambos el primero es menos
    {
      writeblock(mezcla, pedido1);
      readblock(flectura1, pedido1);
   } else
      {
         writeblock(mezcla, pedido2);
         readblock(flectura2, pedido2);
      }
 }
std::cout << totalGeneral <<'\n';
flectura1.close();
flectura2.close();
fmezcla.close();
return 0;
}
```

FILE *	streams
Archivos cabecera #include <stdio.h>	#include <iostream> #include <fstream>

Abrir un archivo
Texto lectura
FILE * f = fopen (“archivo”, “r”);
Texto escritura
FILE * f = fopen (“archivo”, “w”);
Binario lectura
FILE * f = fopen (“archivo”, “rb”);
Binario escritura
FILE * f = fopen (“archivo”, “wb”);
	Ifstream f("archivo",ios::in);
	
	ofstream f("archivo",ios::out);
	
	Ifstream f("archivo",ios::in|| ios::binary);
	
	ofstream f("archivo",ios::out|| ios::binary);

Leer por bloque(restringido a 1 bloque de 1 registro)
fread(&r, sizeof(r), 1, f);
 	
Plantilla Lectura
`template <typename T> T read(FILE* f)`
`{`
`  T buff;`
`  fread(&buff,sizeof(T),1,f);`
`  return buff;`
`}`
	f.read(reinterpret_cast<char*>(&r), sizeof r;
	
	`template<typename T>`
	`std::istream& readblock(std::istream& in, T& block){`
	`  return in.read(`
	`  reinterpret_cast<char*>(&block),`
	`  sizeof block`
	`  );`
	`}`

Grabar por bloque
fwrite(r, sizeof(r),1,f)
 	
Plantilla gabar
`template <typename T> void write(FILE* f, T v)`
`{`
`  fwrite(&v,sizeof(T),1,f);`
`  return;`
`}`
	out.write(reinterpret_cast<const char*>(&r),
	sizeof r)
	
	`template<typename T>`
	`std::ostream& writeblock(std::ostream& out, const T& block){`
	`  return out.write(`
	`  reinterpret_cast<const char*>(&block),`
	`  sizeof block`
	`  );`
	`}`