P

Procesamiento de archivos en C++

Pablo R. Ramis

```cpp
#include <iostream>
int main()
{
    std::cout << "In Code We Trust";
    return 0;
}
```

1. INTRODUCCIÓN

C++ considera a los archivos como una secuencia de bytes. Cada archivo termina con un **marcador de fin de archivo** o un numero de bytes específicos se registra en una estructura de datos administrativa llevada por el sistema. Cuando se *abre* un archivo, se crea un objeto y se asocia un flujo a ese archivo.

Para llevar a cabo el procesamiento de archivos deberemos incluir las cabeceras $< iostream >$ y $< fstream >$ esta última incluye las definiciones de E/S de char (ifsteam, oftream y fstream)

2. ARCHIVOS SECUENCIALES

2.1. Creación de un archivo secuencial

C++ no impone una estructura específica de un archivo, esto quedará en manos del programador que según el sistema que este generando buscará la estructura mas conveniente (guardar un registro determinado)

Una forma simple es haciendolo secuencialmente. Vemos el ejemplo:

```
#include<iostream>
#include <fstream> // flujo de archivo
#include <cstdlib> // funciones de exit
using namespace std;


int main()
{
   // el constructor de ofstream abre el archivo
   ofstream archivoClientesSalida( "clientes.dat", ios::out );
    // sale del programa si no puede crear el archivo
   if  ( !archivoClientesSalida ) // operador ! sobrecargado
   {
       cerr << "No se pudo abrir el archivo" << endl;
       exit( 1 );
   } // fin de if

   cout << "Escriba la cuenta, nombre y saldo." << endl
       << "Escriba fin de archivo para terminar la entrada.\n? ";

   int cuenta;
   char nombre[ 30 ];
   double saldo;

   // lee la cuenta, nombre y saldo de cin, y édespus los coloca en el archivo
   while ( cin >> cuenta >> nombre >> saldo ){
       archivoClientesSalida << cuenta << ' ' << nombre << ' ' << saldo << endl;
       cout << "? ";
   } // fin de while

   return 0; // el destructor de ofstream cierra el archivo
} // fin de main
```

Compilamos y ejecutamos el programa:

```
$ g++ -Wall -osecuencial secuencial.cpp
$ ls
secuencial  secuencial.cpp
$./secuencial
Escriba la cuenta, nombre y saldo.
Escriba fin de archivo para terminar la entrada.
? 2014
Pablo
412.32
? 417
Juan
412.7
? ^D
[1]+  Detenido                ./secuencial
$ ls
clientes.dat  secuencial  secuencial.cpp
$ cat clientes.dat
2014 Pablo 412.32
417 Juan 412.7

$
```

Como vemos, el programa dejo un archivo con el nombre indicado y al mostrar el contenido están los dos registros que se ingresaron por teclado.

Tengamos en cuenta que el archivo se abre exclusivamente para escritura, o sea para salida, por lo tanto se crea un objeto de tipo ***ofstream***

Se pasan dos argumentos al constructor del objeto: el nombre de archivo y el modo de apertura de archivo (línea 10). Para un objeto ofstream, el modo de apertura de archivo puede ser **ios::out** para enviar datos a un archivo, o **ios::app** para adjuntar datos al final de un archivo (sin modificar los datos que ya estén en el archivo). Los archivos existentes que se abren con el modo **ios::out** se truncan o sea se descartan todos los datos en el archivo. Si el archivo especificado no existe todavía, entonces el objeto ofstream crea el archivo, usando ese nombre de archivo.

En la línea 10 se crea un objeto ofstream llamado archivoClientesSalida, asociado con el archivo clientes.dat que se abre en modo de salida. Los argumentos "clientes.dat" e ios::out se pasan al constructor de ofstream, el cual abre el archivo (esto establece una "línea de comunicación" con el archivo).

La siguiente enumeración muestra los diferentes modos de apertura de archivos:

ios::app Añade toda la salida al fi nal del archivo.
ios::ate Abre un archivo en modo de salida y se desplaza hasta el fi nal del archivo (por lo general se utiliza para añadir datos a un archivo). Los datos se pueden escribir en cualquier parte del archivo.
ios::in Abre un archivo en modo de entrada.
ios::out Abre un archivo en modo de salida.
ios::trunc Descarta el contenido del archivo, si es que existe (también es la acción predeterminada para ios::out).
ios::binary Abre un archivo en modo de entrada o salida binaria (es decir, que no es texto).

El programa itera hasta el fin de archivo, $< ctrl - d >$ para linux, $< ctrl - z$ en Windows. Una vez que el usuario introduce el indicador de fin de archivo, main termina. Esto invoca de manera implícita a la función destructor del objeto archivoClientesSalida, que cierra el archivo clientes.dat. Obviamente en un sistema deberíamos dar la opción de un cierre ordenado y explícito usando la función miembro close en la instrucción archivoClientesSalida.close();

2.2. Lectura de un archivo secuencial

En este caso, crearemos un objeto de tipo **ifstream** ya que solo debemos leer la información. Por una cuestión de seguridad, lo recomendado es que cuando hay solo lectura, se haga de ese modo y se evita escrituras accidentales. Veamos un posible código

```
1  #include <iostream>
2  #include <fstream> // flujo de archivo
3  #include <iomanip> // para setw y setprecision
4  #include <string>
5
6  using namespace std;
7
8
9  void imprimirLinea( int, const string, double ); // prototipo
10
11 int main()
12 {
13     // el constructor de ifstream abre el archivo
14     ifstream archivoClientesEntrada( "clientes.dat", ios::in );
15
16     // sale del programa si ifstream no pudo abrir el archivo
17     if ( !archivoClientesEntrada )
18     {
19         cerr << "No se pudo abrir el archivo" << endl;
20         return 1;
21     } // fin de if
22
23     int cuenta;
24     char nombre[ 30 ];
25     double saldo;
26
27     cout << left << setw( 10 ) << "Cuenta" << setw( 13 )
28         << "Nombre" << "Saldo" << endl << fixed << showpoint;
29
30     // muestra cada registro en el archivo
31     while ( archivoClientesEntrada >> cuenta >> nombre >> saldo )
32         imprimirLinea( cuenta, nombre, saldo );
33
34     return 0; // el destructor de ifstream cierra el archivo
35 } // fin de main
36
```

```
// muestra un solo registro del archivo
void imprimirLinea( int cuenta, const string nombre, double saldo ){

    cout << left << setw( 10 ) << cuenta << setw( 13 ) << nombre
        << setw( 7 ) << setprecision( 2 ) << right << saldo << endl;

} // fin de la ófuncin imprimirLinea
```

Compilamos y probamos

```
$ g++ -Wall -olecturaSecuencial lecturaSecuencial.cpp
$ ./lecturaSecuencial
Cuenta     Nombre       Saldo
2014       Pablo         412.32
417        Juan          412.70
$ cat clientes.dat
2014 Pablo 412.32
417 Juan 412.7
$
```

Para obtener datos secuencialmente de un archivo, por lo general los programas empiezan a leer desde el principio del archivo y leen todos los datos en forma consecutiva, hasta encontrar los datos deseados. Tal vez sea necesario procesar el archivo secuencialmente varias veces (desde el principio del mismo) durante la ejecución de un programa. Tanto istream como ostream proporcionan funciones miembro para reposicionar el apuntador de posición del archivo (el número de byte del siguiente byte en el archivo que se va a leer o escribir). Estas funciones miembro son **seekg** (“seek get”, “buscar obtener”) para istream y **seekp** (“seek put”, “buscar colocar”) para ostream. Cada objeto istream tiene un “apuntador obtener”, el cual indica el número de byte en el archivo a partir del cuál va a ocurrir la siguiente entrada, y cada objeto ostream tiene un “apuntador colocar”, el cual indica el número de byte en el archivo en el que se debe colocar la siguiente salida. La instrucción ***archivoClientesEntrada.seekg(0);*** reposiciona el apuntador de posición del archivo y lo coloca al

principio del archivo (ubicación 0) adjunto a archivoClientesEntrada. El argumento para seekg es comúnmente un entero long. Se puede especificar un segundo argumento para indicar la dirección de búsqueda, que puede ser **ios::beg** (la opción predeterminada) para un posicionamiento relativo al inicio de un flujo, **ios::cur** para un posicionamiento relativo a la posición actual en un flujo, o **ios::end** para un posicionamiento relativo al final de un flujo. El apuntador de posición del archivo es un valor entero que especifica la ubicación en el archivo como un número de bytes desde la ubicación inicial del archivo (a ésta también se le conoce como el desplazamiento desde el inicio del archivo). Algunos ejemplos de cómo posicionar el apuntador “obtener” de posición del archivo son:

// se posiciona en el n-ésimo byte de objetoArchivo (asumiendo ios::beg)

objetoArchivo.seekg(n);

// se posiciona n bytes hacia adelante en objetoArchivo

objetoArchivo.seekg(n, ios::cur);

// se posiciona n bytes hacia atrás desde el final de objetoArchivo

objetoArchivo.seekg(n, ios::end);

// se posiciona al final de objetoArchivo

objetoArchivo.seekg(0, ios::end);

Se pueden realizar las mismas operaciones mediante la función miembro seekp de ostream. Las funciones miembro **tellg** y **tellp** se proporcionan para devolver las posiciones actuales de los apuntadores “obtener” y “colocar”, respectivamente. La siguiente instrucción asigna el valor del apuntador ”obtener” de posición del archivo a la variable ubicacion de tipo long:

ubicacion = objetoArchivo.tellg();

Para el siguiente ejemplo, modificaremos el archivo para completar con variedad de datos, con usuarios con saldos negativos, positivos y en cero. El programa, permitirá buscar los saldos segun estos criterios.

```
#include <iostream>
#include <fstream>
#include <iomanip>
```

```
#include <string>
using namespace std;


class banco{
    enum TipoSolicitud { SALDO_CERO = 1, SALDO_CREDITO, SALDO_DEBITO, TERMINAR };
    ifstream archivoClientesSalida;

    int solicitud;
    int cuenta;
    char nombre[ 30 ];
    double saldo;

public:
    banco();
    ~banco(){ archivoClientesSalida.close(); }
    int obtenerSolicitud();
    bool debeMostrar( int, double );
    void imprimirLinea( int, const string, double );
    void lectura();
};


int main()
{
    banco *Banco = new banco();
    Banco->lectura();
    cout << "Fin de ejecucion." << endl;
    delete(Banco);
    return 0; // el destructor de ifstream cierra el archivo
} // fin de main

banco::banco(){
    // el objeto ifstream abre el archivo
    archivoClientesSalida.open( "clientes.dat", ios::in );

    // sale del programa si ifstream no pudo abrir el archivo
    if ( !archivoClientesSalida ){
        cerr << "No se pudo abrir el archivo" << endl;
        exit( 1 );
    } // fin de if
} //fin del constructor

// obtiene la solicitud del usuario
int banco::obtenerSolicitud(){

    cout << "\nEscriba la opcion" << endl
        << " 1 - Listar cuentas con saldos en cero" << endl
        << " 2 - Listar cuentas con saldos de credito" << endl
        << " 3 - Listar cuentas con saldos de debito" << endl
        << " 4 - Finalizar ejecucion" << fixed << showpoint;
    do{
        cout << "\n? ";
        cin >> solicitud;
    } while ( solicitud < SALDO_CERO && solicitud > TERMINAR );
        return solicitud;
} // fin de la ófuncin obtenerSolicitud

// determina si se va a mostrar el registro dado
bool banco::debeMostrar( int tipo, double saldo ){
    // determina si se van a mostrar los saldos en cero
    if ( tipo == SALDO_CERO && saldo == 0 )
        return true;

    // determina si se van a mostrar los saldos de écrdito
    if ( tipo == SALDO_CREDITO && saldo < 0 )
        return true;
    // determina si se van a mostrar los saldos de édbito
    if ( tipo == SALDO_DEBITO && saldo > 0 )
        return true;
```

```
74
75      return false;
76  } // fin de la ófuncin debeMostrar
77
78  // muestra un solo registro del archivo
79  void banco::imprimirLinea( int cuenta, const string nombre, double saldo ){
80      cout << left << setw( 10 ) << cuenta << setw( 13 ) << nombre
81          << setw( 7 ) << setprecision( 2 ) << right << saldo << endl;
82  }// fin de la ófuncin imprimirLinea
83
84  void banco::lectura(){
85      // obtiene la solicitud del usuario (por ejemplo, saldo en cero, de écrdito o édbito)
86      solicitud = obtenerSolicitud();
87
88      // procesa la solcitud del usuario
89      while ( solicitud != TERMINAR ){
90          switch ( solicitud ){
91              case SALDO_CERO:
92                  cout << "\nCuentas con saldos en cero:\n";
93              break;
94              case SALDO_CREDITO:
95                  cout << "\nCuentas con saldos de credito:\n";
96              break;
97              case SALDO_DEBITO:
98                  cout << "\nCuentas con saldos de debito:\n";
99              break;
100         } // fin de switch
101         // lee la cuenta, el nombre y el saldo del archivo
102         archivoClientesSalida >> cuenta >> nombre >> saldo;
103         //muestra el contenido del archivo (hasta eof)
104         while ( !archivoClientesSalida.eof() ){
105             // muestra el registro
106             if ( debeMostrar( solicitud, saldo ) )
107                 imprimirLinea( cuenta, nombre, saldo );
108
109             // lee la cuenta, el nombre y el saldo del archivo
110             archivoClientesSalida >> cuenta >> nombre >> saldo;
111         } // fin de while interior
112
113         archivoClientesSalida.clear(); // restablece eof para la siguiente entrada
114         archivoClientesSalida.seekg( 0 ); // se reposiciona al inicio del archivo
115         solicitud = obtenerSolicitud(); // obtiene una solicitud adicional del usuario
116     } // fin de while exterior
117 }// fin de la funcion lectura
```

Compilamos y probamos

```
$ g++ -Wall -olectura2 lecturaSecuencial_2.cpp
$ ./lectura2

Escriba la opcion
 1 - Listar cuentas con saldos en cero
 2 - Listar cuentas con saldos de credito
 3 - Listar cuentas con saldos de debito
 4 - Finalizar ejecucion
? 1

Cuentas con saldos en cero:
541       Laura           0.00

Escriba la opcion
 1 - Listar cuentas con saldos en cero
 2 - Listar cuentas con saldos de credito
 3 - Listar cuentas con saldos de debito
 4 - Finalizar ejecucion
? 2

Cuentas con saldos de credito:
417       Juan         -412.70

Escriba la opcion
 1 - Listar cuentas con saldos en cero
 2 - Listar cuentas con saldos de credito
 3 - Listar cuentas con saldos de debito
 4 - Finalizar ejecucion
```

```
? 3

Cuentas con saldos de debito:
2014      Pablo         412.32

Escriba la opcion
 1 - Listar cuentas con saldos en cero
 2 - Listar cuentas con saldos de credito
 3 - Listar cuentas con saldos de debito
 4 - Finalizar ejecucion
? 4
Fin de ejecucion.
$
$
```

En el ejemplo, vemos ya una mayor complejidad. Se declara el objeto ifstream pero se inicializa en el constructor, Luego, en la clase, se dividen las funcionalidades en los distintos métodos. El destructor de la clase cierra el archivo.

2.3. Actualización de un archivo secuencial

Si bien es posible hacer modificaciones a los datos de un archivo escrito secuencialmente esto no es recomendable y hasta resulta riesgoso. El ejemplo es simple, si tuvieramos que modificar un apellido, por ejemplo, en el archivo se encuentra "Perez" por "Rodriguez" la diferencia del largo de caracteres compromete y corrompe al archivo por pisar datos contiguos.

3. ARCHIVOS DE ACCESO ALEATORIO

En las aplicaciones que se requiere accesos inmediatos a los registros, esta modalidad de manipulación del archivo es la ideal. Como dijimos antes, C++ no impone un formato, por lo tanto, seremos nosotros los que indicaremos el modo, y estos serán variados. El más simple es requerir que todos los registros sean de una determinada longitud, esto hace que sea más simple calcular las posiciones de los registros como vemos en la

3.1. Creación del archivo aleatorio

La función miembro de ofstream que permite grabar números fijos de bytes es ***write*** y por lo tanto, la función ***read*** será la que corresponda para leer bloques determinados de bytes. Los manejos de los punteros son similares a lo descripto antes.

Por lo tanto, para que un entero de cuatro bytes (desde 1 digito hasta 11, diez más uno para el signo) usaremos la siguiente instrucción:

archivoSalida.write(reinterpret_cast $< const char* >$(& numero), sizeof(numero));

Tengamos en cuenta que habiamos comentado que lo grabado eran chars, como muchas veces no seran const char*, que es lo requerido por la función, necesitamos hacer una conversión de ellos y utilizaremos ***reinterprest_cast***, la conversión se hará en tiempo de ejecución sin cambiar el valor del objeto que es apuntado.

3.1.1. Ejemplo. Siguiendo el ejemplo de las cuentas bancarias, veremos un programa que almacenará registros de longitud fija. Los registros tiene un numero de cuenta como clave, un apellido y un nombre, un saldo. El programa actualizará cuentas, eliminará e insertará registros en un archivo.

```
1  #include <string>
2  #include <cstring>
3  #include <iostream>
4  #include <fstream>
5  #include <cstdlib>
6  using namespace std;
7
8  class DatosCliente
9  {
10     int numeroCuenta;
11     char apellidoPaterno[ 15 ];
12     char primerNombre[ 10 ];
13     double saldo;
14 public:
15     // constructor predeterminado de DatosCliente
16     DatosCliente( int = 0, string = "", string = "", double = 0.0 );
17
18     // funciones de acceso para numeroCuenta
19     void establecerNumeroCuenta( int );
20     int obtenerNumeroCuenta() const;
21
22     // funciones de acceso para apellidoPaterno
23     void establecerApellidoPaterno( string );
```

```
24      string obtenerApellidoPaterno() const;
25
26      // funciones de acceso para primerNombre
27      void establecerPrimerNombre( string );
28      string obtenerPrimerNombre() const;
29
30      // funciones de acceso para el saldo
31      void establecerSaldo( double );
32      double obtenerSaldo() const;
33
34  }; // fin de la clase DatosCliente
35
36  // constructor predeterminado de DatosCliente
37  DatosCliente::DatosCliente( int valorNumeroCuenta, string valorApellidoPaterno, string
        valorPrimerNombre, double valorSaldo )
38  {
39      establecerNumeroCuenta( valorNumeroCuenta );
40      establecerApellidoPaterno( valorApellidoPaterno );
41      establecerPrimerNombre( valorPrimerNombre );
42      establecerSaldo( valorSaldo );
43  } // fin del constructor de DatosCliente
44
45  // obtiene el valor del únmero de cuenta
46  int DatosCliente::obtenerNumeroCuenta() const
47  {
48      return numeroCuenta;
49  } // fin de la ófuncin obtenerNumeroCuenta
50
51  // establece el valor del únmero de cuenta
52  void DatosCliente::establecerNumeroCuenta( int valorNumeroCuenta )
53  {
54      numeroCuenta = valorNumeroCuenta; // debe validar
55  } // fin de la ófuncin establecerNumeroCuenta
56
57  // obtiene el valor del apellido paterno
58  string DatosCliente::obtenerApellidoPaterno() const
59  {
60      return apellidoPaterno;
61  } // fin de la ófuncin obtenerApellidoPaterno
62
63  //establece el valor del apellido paterno
64  void DatosCliente::establecerApellidoPaterno( string cadenaApellidoPaterno )
65  {
66      // copia a lo áms 15 caracteres de la cadena a apellidoPaterno
67      const char *valorApellidoPaterno = cadenaApellidoPaterno.data();
68      int longitud = cadenaApellidoPaterno.size();
69      longitud = ( longitud < 15 ? longitud : 14 );
70      strncpy( apellidoPaterno, valorApellidoPaterno, longitud );
71      apellidoPaterno[ longitud ] = '\0'; // adjunta un ácarcter nulo a apellidoPaterno
72  } // fin de la ófuncin establecerApellidoPaterno
73
74  // obtiene el valor del primer nombre
75  string DatosCliente::obtenerPrimerNombre() const
76  {
77      return primerNombre;
78  } // fin de la ófuncin obtenerPrimerNombre
79
80  // establece el valor del primer nombre
81  void DatosCliente::establecerPrimerNombre( string cadenaPrimerNombre )
82  {
83      // copia a lo áms 10 caracteres de la cadena a primerNombre
84      const char *valorPrimerNombre = cadenaPrimerNombre.data();
85      int longitud = cadenaPrimerNombre.size();
86      longitud = ( longitud < 10 ? longitud : 9 );
87      strncpy( primerNombre, valorPrimerNombre, longitud );
88      primerNombre[ longitud ] = '\0'; // adjunta un ácarcter nulo a primerNombre
89  } // fin de la ófuncin establecerPrimerNombre
90
91  //obtiene el valor del saldo
92  double DatosCliente::obtenerSaldo() const
```

```
{
    return saldo;
} // fin de la ófuncin obtenerSaldo

// establece el valor del saldo
void DatosCliente::establecerSaldo( double valorSaldo )
{
    saldo = valorSaldo;
} // fin de la ófuncin establecerSaldo

int main()
{
    ofstream creditoSalida( "credito.dat", ios::out | ios::binary );

    // sale del programa si ofstream no pudo abrir el archivo
    if ( !creditoSalida ){
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    } // fin de if

    DatosCliente clienteEnBlanco; // el constructor pone en ceros cada miembro de datos

    // escribe 100 registros en blanco en el archivo
    for ( int i = 0; i < 100; i++ )
        creditoSalida.write( reinterpret_cast< const char * >( &clienteEnBlanco ), sizeof
            ( DatosCliente ) );
    return 0;
} // fin de main
```

Se crea un objeto ofstream para el archivo credito.dat. El segundo argumento para el constructor (ios::out | ios::binary) indica que vamos a abrir el archivo para salida en modo binario, lo cual es requerido si debemos escribir registros de longitud fija.

A modo de ejemplo, la escritura fue de un objeto el cual no poseía sus atributos inicializados, lo cual en este caso no es trascendente ya que el espacio se ocupará igual. El contenido del archivo no será visible, no solo porque los objetos no estaban seteados, sino porque están grabajos en forma binaria.

3.2. Escribir datos al azar en un archivo de acceso aleatorio

Para escribir datos en el archivo **credito.dat** utilizaremos la función ***seekp*** y ***write*** de fstream. La primera establece la posición del puntero en un lugar específico para escribir, y la segunda ya fue explicado, es la encargada de grabar.

El código siguiente es solo el main, el cual reemplazaría al del ejemplo anterior.

```
// El ócdigo es éidntico al ejemplo anterior

int main()
{

    int numeroCuenta;
    char apellidoPaterno[ 15 ];
    char primerNombre[ 10 ];
    double saldo;

    fstream creditoSalida( "credito.dat", ios::in | ios::out | ios::binary );

    // sale del programa si fstream no puede abrir el archivo
    if ( !creditoSalida ){
        cerr << "No se pudo abrir el archivo." << endl;
        exit( 1 );
    } // fin de if

    cout << "Escriba el numero de cuenta (de 1 a 100, 0 para terminar la entrada)\n? ";
    // requiere que el usuario especifique el únmero de cuenta
    DatosCliente cliente;
    cin >> numeroCuenta;

    // el usuario introduce óinformacin, la cual se copia en el archivo
    while ( numeroCuenta > 0 && numeroCuenta <= 100 ){
        // el usuario introduce el apellido paterno, primer nombre y saldo
        cout << "Escriba apellido paterno, primer nombre y saldo\n? ";
```

```
28          cin >> setw( 15 ) >> apellidoPaterno;
29          cin >> setw( 10 ) >> primerNombre;
30          cin >> saldo;
31
32          //establece los valores de numeroCuenta, apellidoPaterno, primerNombre y saldo
                del registro
33          cliente.establecerNumeroCuenta( numeroCuenta );
34          cliente.establecerApellidoPaterno( apellidoPaterno );
35          cliente.establecerPrimerNombre( primerNombre );
36          cliente.establecerSaldo( saldo );
37
38          // busca la óposicin en el archivo del registro especificado por el usuario
39          creditoSalida.seekp( ( cliente.obtenerNumeroCuenta() - 1 ) * sizeof( DatosCliente
                ) );
40
41          // escribe la óinformacin especificada por el usuario en el archivo
42          creditoSalida.write( reinterpret_cast< const char * >( &cliente ), sizeof(
                DatosCliente ) );
43
44          // permite al usuario escribir otro únmero de cuenta
45          cout << "Escriba el numero de cuenta\n? ";
46          cin >> numeroCuenta;
47      } // fin de while
48
49      return 0;
50
51  } // fin de main
```

compilamos y corremos.

```
$
Escriba el numero de cuenta (de 1 a 100, 0 para terminar la entrada)
? 54
Escriba apellido paterno, primer nombre y saldo
? Ramis Pablo 124.21
Escriba el numero de cuenta
? 41
Escriba apellido paterno, primer nombre y saldo
? Diaz Julian -9521.0
Escriba el numero de cuenta
? 32
Escriba apellido paterno, primer nombre y saldo
? Suarez Jorgelina 0
Escriba el numero de cuenta
? 0
$
```

El programa aunque incompleto situa la posición en el registro que posee el numero de cuenta bancaria. Recordamos que se habian guardado 100 registros en blanco. Aquí se pide el numero de cuenta de 1 a 100 y con ***seekp*** se posiciona en él como se ve en la línea 39.

3.3. Leer un archivo de acceso aleatorio

Para la lectura utilizaremos la función ***read***, el proceso es similar a lo visto, pero con resultados opuestos . La función debe recibir un primer argumento de tipo $char*$ por lo tanto se deberá convertir del mismo modo que antes: con la función reinterpret_cast.

En el ejemplo, la clase y los prototipos son idénticos, solo cambiaremos el main y agregaremos imprimeLinea

```
1   // El ócdigo es éidntico al ejemplo anterior
2
3   void imprimirLinea( ostream&, const DatosCliente & ); // prototipo
4
5   int main()
6   {
7       ifstream creditoEntrada( "credito.dat", ios::in | ios::binary );
8
9       // sale del programa si ifstream no puede abrir el archivo
10      if ( !creditoEntrada ) {
11          cerr << "No se pudo abrir el archivo." << endl;
12          exit( 1 );
13      } // fin de if
14
```

```
15      cout << left << setw( 10 ) << "Cuenta" << setw( 16 )
16          << "Apellido" << setw( 11 ) << "Nombre" << left
17          << setw( 10 ) << right << "Saldo" << endl;
18
19      DatosCliente cliente; // crea un registro
20
21      // lee el primer registro del archivo
22      creditoEntrada.read( reinterpret_cast< char * >( &cliente ),sizeof( DatosCliente ) );
23
24      // lee todos los registros del archivo
25      while ( creditoEntrada && !creditoEntrada.eof() ){
26          // muestra un registro
27          if ( cliente.obtenerNumeroCuenta() != 0 )
28              imprimirLinea( cout, cliente );
29
30          // lee el siguiente registro del archivo
31          creditoEntrada.read( reinterpret_cast< char * >( &cliente ), sizeof( DatosCliente
                ) );
32      } // fin de while
33
34      return 0;
35  } // fin de main
36
37  // muestra un solo registro
38  void imprimirLinea( ostream &salida, const DatosCliente &registro ){
39      salida << left << setw( 10 ) << registro.obtenerNumeroCuenta()
40          << setw( 16 ) << registro.obtenerApellidoPaterno()
41          << setw( 11 ) << registro.obtenerPrimerNombre()
42          << setw( 10 ) << setprecision( 2 ) << right << fixed
43          << showpoint << registro.obtenerSaldo() << endl;
44  } // fin de la ófuncin imprimirLinea
```

compilamos y corremos.

```
$g++ -Wall -olecturaAleatoria lecturaAleatoria.cpp
$ ./lecturaAleatoria
Cuenta    Apellido        Nombre          Saldo
32        Suarez          Jorgelina        0.00
41        Diaz            Julian       -9521.00
54        Ramis           Pablo          124.21
$
$
```

El código no representa un desafío, recorre el archivo leyendo de a bloques del tamaño de la estructura que se guardó. Aquellas que no son vacias no se tienen en cuenta, las que sí se asignan a un objeto de tipo cliente y se muestran por pantalla.

4. EJEMPLO

Incorporaremos un ejemplo completo del grafo, que hemos visto en temas anteriores, con la opción de cargarlo desde un archivo y grabarlo en uno. Para mejor comprensión, el codigo completo será transcripto.

```
1   #include<iostream>
2   #include<fstream>
3   #include<vector>
4   #include<utility>
5   using namespace std;
6
7   /*
8   Sea un grafo G = (V, E)
9   */
10  class grafo{
11
12          vector<char> V;
13          vector<pair<pair<char, char>, int>> E;
14          const string nombre_archivo = "archivo.dat";
15
16  public:
17          grafo(){}
18          ~grafo(){}
19
```

```
20          void insertar_vertice(const char&);
21          void insertar_arista();
22
23          void escribe_grafo();
24          void lee_grafo();
25
26          friend ostream& operator <<(ostream&, grafo);
27  };
28
29  void grafo::insertar_vertice(const char& vertice){
30          V.push_back(vertice);
31  }
32
33  void grafo::insertar_arista(){
34          int c = 0;
35          for(unsigned int x = 0; x <= V.size()-1; x++){
36                  for(unsigned int j = x+1; j <= V.size()-1; j++){
37                          cout << "Ingrese costo para arista entre vertice " << V[x] << " y
                                 " << V[j]
38                             << " (0 si no existe arista): ";
39                          cin >> c;
40                          cout << endl;
41                          if (c != 0){
42                                  E.push_back(make_pair(make_pair(V[x], V[j]), c));
43                          }
44                  }
45          }
46  }
47
48  void grafo::escribe_grafo(){
49          ofstream archivo( nombre_archivo, ios::out | ios::binary );
50
51          // sale del programa si ofstream no pudo abrir el archivo
52      if ( !archivo ){
53          cerr << "No se pudo abrir el archivo." << endl;
54          exit( 1 );
55      } // fin de if
56
57          for(auto x: E)
58                  archivo << x.first.first << x.first.second << x.second;
59
60          archivo.close();
61
62  }
63
64  void grafo::lee_grafo(){
65          ifstream archivo(nombre_archivo, ios::in | ios::binary );
66           // sale del programa si ifstream no puede abrir el archivo
67      if ( !archivo ) {
68          cerr << "No se pudo abrir el archivo." << endl;
69          exit( 1 );
70      } // fin de if
71
72      char nodoOrigen, nodoDestino;
73      int costo;
74
75          cout<< "Datos del archivo\n";
76      archivo >> nodoOrigen >> nodoDestino >> costo;
77      while(archivo && !archivo.eof()){
78          cout << nodoOrigen << " - " << nodoDestino << ": costo = " << costo << endl;
79          archivo >> nodoOrigen >> nodoDestino >> costo;
80      }
81      cout << nodoOrigen << " - " << nodoDestino << ": costo = " << costo << endl;
82      archivo.close();
83
84  }
85
86  ostream& operator <<(ostream& os, grafo G){
87
88          os << "El grafo es: \n";
```

```
89          for(auto x: G.E)
90                  os << x.first.first << " - " << x.first.second << ": costo = " << x.
                        second << endl;
91
92          return os;
93  }
94
95  int main(){
96          grafo G;
97          char x;
98          cout << "Ingrese las letras que representan a los vertices (0 cuando finalice)";
99          cin >> x;
100         while(x != '0'){
101                 G.insertar_vertice(x);
102                 cin >> x;
103         }
104
105         G.insertar_arista();
106         G.escribe_grafo();
107
108         G.lee_grafo();
109
110         cout << G;
111         return 0;
112 }
```

compilamos y corremos.

```
$g++ -Wall -std=c++11 -ografo grafo_file.cpp
$ ./grafo
Ingrese las letras que representan a los vertices (0 cuando finalice)a
b
c
d
e
0
Ingrese costo para arista entre vertice a y b (0 si no existe arista): 1

Ingrese costo para arista entre vertice a y c (0 si no existe arista): 2

Ingrese costo para arista entre vertice a y d (0 si no existe arista): 3

Ingrese costo para arista entre vertice a y e (0 si no existe arista): 0

Ingrese costo para arista entre vertice b y c (0 si no existe arista): 9

Ingrese costo para arista entre vertice b y d (0 si no existe arista): 0

Ingrese costo para arista entre vertice b y e (0 si no existe arista): 1

Ingrese costo para arista entre vertice c y d (0 si no existe arista): 3

Ingrese costo para arista entre vertice c y e (0 si no existe arista): 0

Ingrese costo para arista entre vertice d y e (0 si no existe arista): 4

Datos del archivo
a - b: costo = 1
a - c: costo = 2
a - d: costo = 3
b - c: costo = 9
b - e: costo = 1
c - d: costo = 3
d - e: costo = 4
El grafo es:
a - b: costo = 1
a - c: costo = 2
a - d: costo = 3
b - c: costo = 9
b - e: costo = 1
c - d: costo = 3
d - e: costo = 4
$
$cat archivo.dat
ab1ac2ad3bc9be1cd3de4$
$
```