Arquivos

BCC 221 - Programação Orientada a Objectos(POO)

Guillermo Cámara-Chávez

Departamento de Computação - UFOP

Introdução

- O armazenamento em variáveis e vetores é temporário;
  - Arquivos são utilizados para persistência de dados
- Uma maneira comum de organizar dados em arquivos é a sequencial
  - Grupos de arquivos relacionados são frequentemente armazenados em bancos de dados.

Introdução (cont.)

- Em C++, um arquivo é uma sequência de *bytes*
  - O final de cada arquivo é indicado pelo **marcador de fim de arquivo**
  - Quando um arquivo é aberto, um objeto é criado e um fluxo de dados associado a ele.

Introdução (cont.)

- C++ fornece três classes para lidar com arquivos:
  - **ofstream**: para escrever em arquivos ("o" = *output*);
  - **ifstream**: para ler arquivos ("i" = *input*);
  - **fstream**: para ler e/ou escrever em arquivos.
- O uso destas classes é bastante similar ao uso de **cin** e **cout**.
- Para trabalhar com essas classes precisamos usar a biblioteca `<fstream>`
- *Stream*: é uma seqüência de bytes

Classe *fstream*

- Declaração

```
fstream arquivo;
```

- Abrir arquivo só para leitura:

```
arquivo.open("meuArquivo.txt", ios::in);
```

- Abrir arquivo só para escrita:

```
arquivo.open("meuArquivo.txt", ios::out);
```

- Abrir arquivo para escrita e leitura:

```
arquivo.open("meuArquivo.txt", ios::in|ios::out
    );
```

Modos de abertura de arquivos

- Especifica como o arquivo deve ser **aberto** e o que **pode ser feito** com ele
- `ios::in` e `ios::out` são exemplos de modos de abertura de arquivos
- *Flags* de abertura de arquivo podem ser combinados na abertura

Modos de abertura de arquivos (cont.)

- *Flags* de abertura

`ios::app`	Cria novo arquivo, ou adiciona ao final de um arquivo existente
`ios::ate`	Vai para o final do arquivo; escreve em qualquer lugar
`ios::binary`	Lê/escreve em modo binário
`ios::in`	Abre para leitura
`ios::out`	Abre para escrita

Modos de abertura de arquivos (cont.)

- `ifstream` e `ofstream` possuem modos *default*
- O **segundo parâmetro** da função `open` é **opcional** quando utilizado `ifstream` e `ofstream`

Modos *Default*

- `ofstream`:
  - Abertura somente para escrita
  - Não são permitidas leituras
  - Se não existe o arquivo é criado
  - O conteúdo é apagado caso o arquivo exista

Modos *Default* (cont.)

- `ifstream`:
  - Abertura somente para leitura
  - Não é permitido escrever no arquivo
  - A abertura falha caso o arquivo não exista

Escrita e Leitura

- Leitura e Escrita simultânea em arquivos
  - Leitura de dados do arquivo: memória
  - Atualização de dados
  - Escrita dos dados atualizados no arquivo

Arquivo de Entrada

```
#include <fstream>
int main(){
    ...
    ifstream arqIn;
    ...
}
```

- `arqIn.open("nomeArquivo.txt");`
  - Conecta a *stream arqIn* ao arquivo `"nomeArquivo.txt"`.
- `arqIn.close()`
  - Desconecta a *stream* do arquivo associado.
- `arqIn >> c;`
  - Comportamento idêntico ao `cin`.

Arquivo de Saída

```
#include <fstream>
int main(){
    ...
    ofstream arqOut;
    ...
}
```

- `arqOut.open("nomeArquivo.txt");`
  - Conecta a *stream arqOut* ao arquivo `"nomeArquivo.txt"`.
- `arqOut.close()`
  - Desconecta a *stream* do arquivo associado.
- `arqOut << c;`
  - Comportamento idêntico ao `cout`.

Escrita

- Vejamos um exemplo de um programa que lê um número de conta, o nome de um cliente e seu saldo em relação a uma empresa de crédito
  - Utilizaremos um arquivo sequencial de somente escrita
  - O programa supõe que o usuário digitará os três dados sempre na mesma ordem

Escrita (cont.)

```
#include <iostream>
#include <fstream> // fluxo de arquivo
#include <cstdlib>
using namespace std;

int main()
{
    // construtor ofstream abre arquivo
    ofstream outClientFile("clients.dat", ios::out);

    // fecha o programa se não conseguir criar arquivo
    if (!outClientFile) // operador ! sobrecarregado
    {
        cerr << "File could not be opened" << endl;
        exit(1);
    } // fim do if

    cout << "Enter the account, name, and balance.\n"
        << "Enter end-of-file to end input.\n? ";
```

Escrita (cont.)

```
    int account;
    char name[30];
    double balance;

    // lê conta, nome e saldo a partir de cin, então coloca no arquivo
    while (cin >> account >> name >> balance)
    {
        outClientFile << account << ' ' << name << ' '
            << balance << endl;
        cout << "? ";
    } // fim do while
    outClientFile.close();
    return 0; // destrutor ofstream fecha o arquivo
} // fim de main
```

Escrita (cont.)

```
Informe a conta, o nome e o saldo.
Fim de arquivo para terminar a entrada
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Escrita (cont.)

Para finalizar a execução

Sistema Operacional	**Combinação do Teclado**
Unix/Linux/Mac OS X	<ctrl+d> (em uma linha isolada
Windows	<ctrl+z> (algumas vezes seguido de *enter*)

Arquivos

- Um objeto *ofstream* pode ser criado sem ser associado a um arquivo
  - Posteriormente, podemos utilizar o método *open()* para associar o objeto a um arquivo

```
ofstream outClientFile;
outClientFile.open("clients.dat", ios::out);
```

Arquivos (cont.)

- O destrutor de um objeto *ofstream* fecha o arquivo
  - No entanto, é possível fechar o arquivo explicitamente, utilizando o método *close()*

```
outClientFile.close();
```

Leitura

```
#include <iostream>
#include <fstream> // fluxo de arquivo include ¡iomanip¿
#include <string>
#include <cstdlib> // exit: sai do protótipo de função

using namespace std;

void outputLine(int, const string, double); // protótipo

int main() {
 // construtor ifstream abre o arquivo
    ifstream inClientFile("clients.dat", ios::in);
    // fecha o programa se ifstream não pôde abrir o arquivo
    if (!inClientFile) {
        cerr << "File could not be opened" << endl;
        exit(1);
    } // fim do if
```

Leitura (cont.)

```
    int account;
    char name[30];
    double balance;

    cout << left << setw(10) << "Account" << setw(13)
         << "Name" << "Balance" << endl << fixed
         << showpoint;

    // exibe cada registro no arquivo
    while (inClientFile >> account >> name >> balance)
        outputLine(account, name, balance);

    return 0; // destrutor ifstream fecha o arquivo
} // fim de main
```

Leitura (cont.)

```
// exibe um registro do arquivo
void outputLine(int account, const string name, double
    balance)
{
    cout << left << setw(10) << account << setw(13)
         << name << setw(7) << setprecision(2) << right
         << balance << endl;
} // fim da função outputLine
```

Leitura (cont.)

Conta	Nome	Saldo
100	Jones	24.98
200	Doe	345.67
300	White	0.00
400	Stone	−42.16
500	Rich	224.62

Leitura (cont.)

- A leitura sequencial do arquivo é realizada pelo laço *while*
  - Quando o final do arquivo for atingido, será retorando **null**, que é convertido para **false** e termina o laço

Ponteiro de Posição

- Para **ler sequencialmente** de um arquivo, os programas normalmente **começam do início**, e lêem os dados até o final do arquivo
  - **Pode ser necessário processar** sequencialmente **várias vezes** um mesmo arquivo
  - Ambos *istream* e *ostream* fornecem métodos para reposicionar o **ponteiro de posição** no arquivo a ser lido ou escrito

Ponteiro de Posição (cont.)

Ponteiro	Classe	Tipo	Descrição
seekg (seek get)	*istream*	*get pointer*	Indica a posição (em *bytes*) do arquivo em que o próximo dado será lido.
seekp (seek put)	*ostream*	*put pointer*	Indica a posição (em *bytes*) do arquivo em que o próximo dado será escrito.

Ponteiro de Posição (cont.)

- Método *seekg*

```
istream& seekg(streampos off, ios_base::seekdir
    way)
```

  - **off**: quantidade de bytes para se deslocar.
  - **way**: posição de início do deslocamento (`ios::beg`, `ios::cur`, `ios::end`).

- As mesmas operações podem ser realizadas utilizando-se o método *seekp* da classe *ostream*

Ponteiro de Posição (cont.)

```
// posiciona no n-ésimo byte do arquivo (assume ios::beg)
fileObject.seekg( n );

// posiciona n bytes à frente da posição atual
fileObject.seekg( n, ios::cur );

// posiciona n bytes antes do fim do arquivo
fileObject.seekg( n, ios::end );

// posiciona no fim do arquivo
fileObject.seekg( 0, ios::end );
```

Ponteiro de Posição (cont.)

- Os métodos **tellg** e **tellp** retornam a posição atual dos pointeiros **get** e **put**, respectivamente

```
long location = fileObject.tellg();
```

Exemplo

Criar um programa que permite ao Gerente de crédito exibir as informações de conta daqueles clientes com saldo zero, saldo credor (negativo) e saldo devedor (positivo). O programa deve exibir um menu que permite selecionar uma das três opções de crédito e uma quarta opção para finalizar.

Exemplo (cont.)

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

enum RequestType { ZERO_BALANCE = 1, CREDIT_BALANCE,
    DEBIT_BALANCE, END };
int getRequest();
bool shouldDisplay(int, double);
void outputLine(int, const string, double);

int main()
{
    // construtor ifstream abre o arquivo
    ifstream inClientFile("clients.dat", ios::in);
```

Exemplo (cont.)

```
// fecha o programa se ifstream não pôde abrir o arquivo
if (!inClientFile)
{
     cerr << "File could not be opened" << endl;
     exit(1);
} // fim do if

int request;
int account;
char name[30];
double balance;

// obtém a solicitação do usuário (por exemplo, saldo zero, credor ou
     devedor)
request = getRequest();
```

Exemplo (cont.)

```
// processa solicitação do usuário
while (request != END){
    switch (request){
    case ZERO_BALANCE:
        cout << "\nAccounts with zero balances:\n";
        break;
    case CREDIT_BALANCE:
        cout << "\nAccounts with credit balances:\n"
            ;
        break;
    case DEBIT_BALANCE:
        cout << "\nAccounts with debit balances:\n";
        break;
    } // fim do switch
    // lê a conta, nome e saldo do arquivo
    inClientFile >> account >> name >> balance;
```

Exemplo (cont.)

```
            // exibe conteúdo do arquivo (até eof)
            while (!inClientFile.eof()){
                // exibe o registro
                if (shouldDisplay(request, balance))
                    outputLine(account, name, balance);

                // lê a conta, nome e saldo do arquivo
                inClientFile >> account >> name >> balance;
            } // fim do while interno
            // redefine eof para próxima entrada
            inClientFile.clear();
            // reposiciona no começo de arquivo
            inClientFile.seekg(0);
            // obtém solicitação adicional do usuário
            request = getRequest();
        } // fim do while externo

        cout << "End of run." << endl;
        return 0; // destrutor ifstream fecha o arquivo
    } // fim de main
```

Exemplo (cont.)

```
int getRequest() // obtém a solicitação do usuário
{
    int request; // solicitação do usuário
    // exibe opções de solicitação
    cout << "\nEnter request" << endl
        << " 1 - List accounts with zero balances" <<
            endl
        << " 2 - List accounts with credit balances" <<
             endl
        << " 3 - List accounts with debit balances" <<
            endl
        << " 4 - End of run" << fixed << showpoint;

    do // entrada da solicitação do usuário
    {
        cout << "\n? ";
        cin >> request;
    } while (request < ZERO_BALANCE && request > END);

    return request;
} // fim da função getRequest
```

Exemplo (cont.)

```
// determina se exibe um dado registro
bool shouldDisplay(int type, double balance)
{
    // determina se exibe saldos zero
    if (type == ZERO_BALANCE && balance == 0)
        return true;

    // determina se exibe saldos de crédito
    if (type == CREDIT_BALANCE && balance < 0)
        return true;

    // determina se exibe saldos de débito
    if (type == DEBIT_BALANCE && balance > 0)
        return true;

    return false;
} // fim da função shouldDisplay
```

Exemplo (cont.)

```
// exibe um registro do arquivo
void outputLine(int account, const string name, double
    balance)
{
    cout << left << setw(10) << account << setw(13)
        << name << setw(7) << setprecision(2)
        << right << balance << endl;
} // fim da função outputLine
```

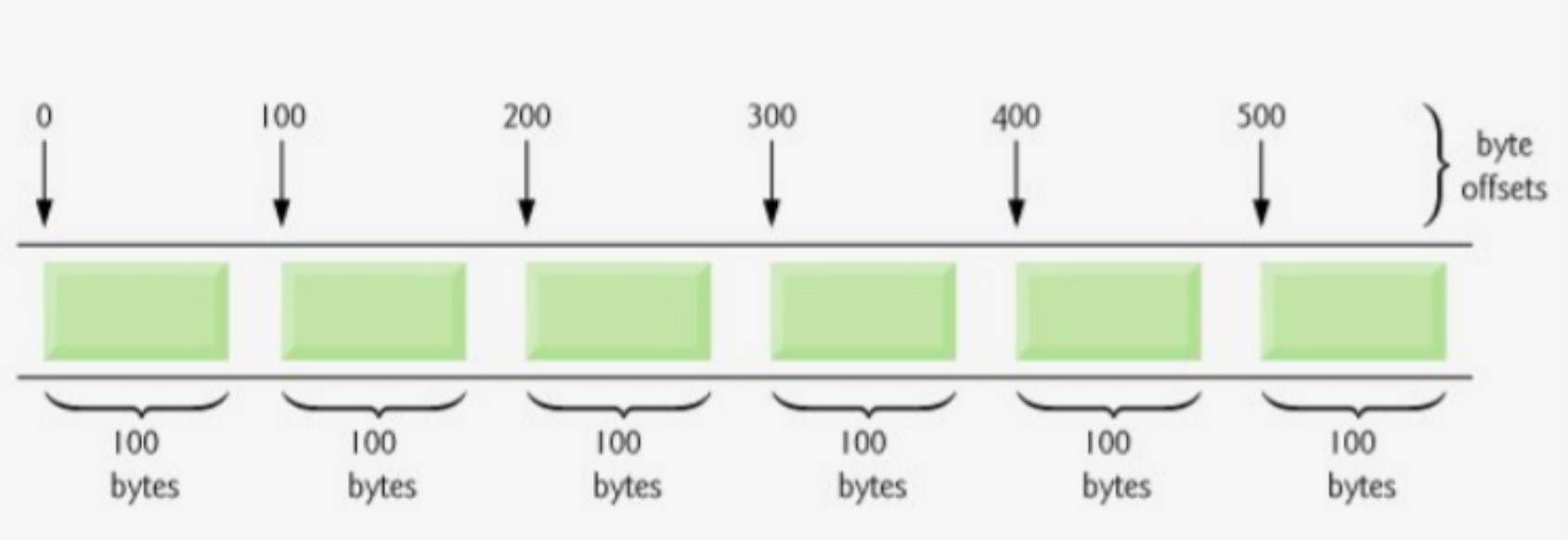
Arquivos de Acesso Aleatório

- C++ não impõe um formato fixo para os arquivos
  - Logo, o acesso instantâneo não é adequado
- Quaisquer aplicações que exijam este tipo de acesso (normalmente sistemas de processamento de transações) devem utilizar **arquivos de acesso aleatório**

Arquivos de Acesso Aleatório (cont.)

- A aplicação deve criar um formato fixo para o arquivo
  - Como por exemplo, obrigar que todos os campos de um arquivo tenham tamanhos fixos
  - Desta forma, é fácil determinar quantos dados serão "pulados" em uma operação.

Arquivos de Acesso Aleatório (cont.)



Arquivos de Acesso Aleatório (cont.)

- **Dados podem ser inseridos** em arquivos de acesso aleatório **sem destruir os outros dados** dos arquivo
- **Dados anteriores também podem ser atualizados ou removidos** sem a necessidade de reescrever todo o arquivo

Arquivos de Acesso Aleatório (cont.)

- O método **write()** da classe **ostream()** escreve um número fixo de *bytes*, a partir de uma determinada posição
  - Grava na posição determinado pelo ponteiro de posição *put*
- O método **read()** da classe **istream()** lê um número fixo de *bytes*, a partir de uma determinada posição
  - Grava na posição determinado pelo ponteiro de posição *get*

Escrita (Modo Binário)

```
ostream& write(reinterpret_cast<const char*>, int);
```

- primeiro parâmetro: é o endereço de memória onde se encontram os dados
- segundo parâmetro: o tamanho em bytes do dado a ser escrito

Leitura (Modo Binário)

```
istream& read(reinterpret_cast<char*>, int);
```

- primeiro parâmetro: é o endereço de memória em que vai ser armazenado o que for lido
- segundo parâmetro: o tamanho em bytes do dado a ser lido

Arquivos de Acesso Aleatório

- Claramente, na maioria das vezes, o primeiro parâmetro não será um ponteiro para **const char**
  - Desta forma, é necessário realizar a conversão através do operador **reinterpret_cast**
  - converte o tipo de ponteiro, não do valor apontado

Exemplos

- Nos exemplos a seguir, considera-se o seguinte contexto:
  - Um programa deve ser capaz de armazenar até 100 registros de tamanho fixo para uma companhia que pode ter até 100 clientes
  - Cada registro consiste de um número de conta (que serve como chave), sobrenome, primeiro nome e saldo
  - O programa deve ser capaz de atualizar uma conta, inserir uma nova conta, remover uma conta e imprimir todas as contas em um arquivo de texto formatado

Exemplos (cont.)

- Veremos 3 exemplos
  - O primeiro mostra as definições da classe utilizada e um *driver* simples, que escreve o conteúdo de um objeto em um arquivo binário de acesso aleatório
  - O segundo escreve os dados para um arquivo e utiliza os métodos **seekp** e **write** para armazenar os dados em posições específicas do arquivo
  - O terceiro lê os dados do arquivo de acesso aleatório e imprime somente os registros que contém dados

ClientData.h

```
#ifndef CLIENTDATA_H
#define CLIENTDATA_H

#include <string>
using std::string;

class ClientData
{
public:
   // construtor ClientData padrão
   ClientData( int = 0, string = "", string = "",
      double = 0.0 );

   // funções de acesso para accountNumber
   void setAccountNumber( int );
   int getAccountNumber() const;
```

ClientData.h (cont.)

```
    // funções de acesso para lastName
    void setLastName( string );
    string getLastName() const;

    // funções de acesso para firstName
    void setFirstName( string );
    string getFirstName() const;

    // funções de acesso para balance
    void setBalance( double );
    double getBalance() const;
private:
    int accountNumber;
    char lastName[ 15 ];
    char firstName[ 10 ];
    double balance;
}; // fim da classe ClientData

#endif
```

ClientData.cpp

```
#include <string>
using std::string;
#include "ClientData.h"

// construtor ClientData padrão
ClientData::ClientData( int accountNumberValue,
    string lastNameValue, string firstNameValue, double
        balanceValue )
{
    setAccountNumber( accountNumberValue );
    setLastName( lastNameValue );
    setFirstName( firstNameValue );
    setBalance( balanceValue );
} // fim do construtor ClientData

// obtém o valor do número da conta
int ClientData::getAccountNumber() const
{
    return accountNumber;
} // fim da função getAccountNumber
```

ClientData.cpp (cont.)

```
// configura o valor do número da conta
void ClientData::setAccountNumber( int
    accountNumberValue ){
   accountNumber = accountNumberValue; // deve validar
} // fim da função setAccountNumber

// obtém o valor do sobrenome
string ClientData::getLastName() const{
   return lastName;
} // fim da função getLastName

// configura o valor do sobrenome
void ClientData::setLastName( string lastNameString ){
   // copia no máximo 15 caracteres da string para lastName
   const char *lastNameValue = lastNameString.data();
   int length = lastNameString.size();
   length = ( length < 15 ? length : 14 );
   strncpy( lastName, lastNameValue, length );
   lastName[ length ] = '\0'; // acrescenta caractere nulo ao
       sobrenome
} // fim da função setLastName
```

ClientData.cpp (cont.)

```
// obtém o valor do nome
string ClientData::getFirstName() const
{
   return firstName;
} // fim da função getFirstName

// configura o valor do nome
void ClientData::setFirstName( string firstNameString )
{
   // copia no máximo 10 caracteres da string para firstName
   const char *firstNameValue = firstNameString.data();
   int length = firstNameString.size();
   length = ( length < 10 ? length : 9 );
   strncpy( firstName, firstNameValue, length );
   firstName[ length ] = '\0'; // acrescenta o caractere nulo a
       firstName
} // fim da função setFirstName
```

ClientData.cpp (cont.)

```
// obtém o valor do saldo
double ClientData::getBalance() const
{
   return balance;
} // fim da função getBalance

// configura o valor do saldo
void ClientData::setBalance( double balanceValue )
{
   balance = balanceValue;
} // fim da função setBalance
```

Exemplo 1 - driverClientData.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include "ClientData.h" // Definição da classe ClientData

int main()
{
   ofstream outCredit( "credit.dat", ios::binary );

   // fecha o programa se ofstream não pôde abrir o arquivo
   if ( !outCredit )
   {
      cerr << "File could not be opened." << endl;
      exit( 1 );
   } // fim do if
```

Exemplo 1 - driverClientData.cpp (cont.)

```
   ClientData blankClient; // construtor zera, ou apaga, cada
       membro de dados

   // gera a saída de 100 registros em branco no arquivo
   for ( int i = 0; i < 100; i++ )
      outCredit.write(
         reinterpret_cast<const char *>( &blankClient ),
         sizeof( ClientData ) );

   return 0;
} // fim de main
```

Exemplo 1 - driverClientData.cpp (cont.)

- Nesta exemplo, 100 objetos vazios foram escritos no arquivo:
  - O número da conta é sempre 0
  - Os nomes são as *string* vazia “”
  - saldo 0.0
- Cada registro é inicializado com a quantidade de espaços vazios correspondente aos dados armazenados
- O programa trata o tamanho de cada *string* para evitar que exceda o tamanho do campo a ser escrito

Exemplo 2 - driverClientData.cpp

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include "ClientData.h" // definição da classe ClientData
using namespace std;
int main(){
   int accountNumber;
   char lastName[ 15 ];
   char firstName[ 10 ];
   double balance;

   fstream outCredit( "credit.dat", ios::in | ios::out
      | ios::binary );

   // sai do programa se fstream não puder abrir o arquivo
   if ( !outCredit )   {
      cerr << "File could not be opened." << endl;
      exit( 1 );
   } // fim do if
```

Exemplo 2 - driverClientData.cpp (cont.)

```
cout << "Enter account number (1 to 100, 0 to end
    input)\n? ";

// requer que usuário especifique o número da conta
ClientData client;
cin >> accountNumber;

// o usuário insere informações, que são copiadas para o arquivo
while (accountNumber > 0 && accountNumber <= 100){
    // o usuário insere o sobrenome, o nome e o saldo
    cout << "Enter lastname, firstname, balance\n? ";
    cin >> setw( 15 ) >> lastName;
    cin >> setw( 10 ) >> firstName;
    cin >> balance;

    // configura valores de accountNumber, lastName, firstName e
        balance
    client.setAccountNumber( accountNumber );
    client.setLastName( lastName );
    client.setFirstName( firstName );
    client.setBalance( balance );
```

Exemplo 2 - driverClientData.cpp (cont.)

```
      // busca posição no arquivo de registro especificado pelo usuário
      outCredit.seekp( (client.getAccountNumber() - 1)
         * sizeof( ClientData ) );

      // grava as informações especificadas pelo usuário no arquivo
      outCredit.write(
        reinterpret_cast<const char*>( &client ),
        sizeof( ClientData ) );

      // permite ao usuário inserir outra conta
      cout << "Enter account number\n? ";
      cin >> accountNumber;
   } // fim do while

   return 0;
} // fim de main
```

Exemplo 2 - driverClientData.cpp (cont.)

```
Enter account number (1 to 100, 0 to end input)
? 37
Enter lastname, firstname, balance
? Barker      Doug          0.00
Enter account number
? 29
Enter lastname, firstname, balance
Brown      Nancy      -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone       Sam          34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith       Dave        258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn        Stacey      314.33
```

Exemplo 2 - driverClientData.cpp (cont.)

- Note que o arquivo foi aberto para leitura e escrita em binário
  - Diferentes modos de abertura podem ser combinados na abertura do arquivo, separados por |
- Cada conta possui uma posição predeterminada no arquivo
  - A conta 1 é a primeira
  - A conta 100 é a última

Exemplo 2 - driverClientData.cpp (cont.)

- Uma conta não é colocada fora de sua posição
  - Mesmo que as contas 1 e 2 não existam, a conta 3 é colocada na terceira posição.
- O ponteiro *put* é posicionado no arquivo de acordo com o número da conta

```
( client.getAccountNumber() - 1 ) *
          sizeof( ClientData )
```

Exemplo 3 - driverClientData.cpp

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
#include "ClientData.h" // definição da classe ClientData

void outputLine( ostream&, const ClientData & ); //
    protótipo

int main()
{
   ifstream inCredit( "credit.dat", ios::in );

   // fecha o programa se ifstream não puder abrir o arquivo
   if ( !inCredit )
   {
      cerr << "File could not be opened." << endl;
      exit( 1 );
   } // fim do if
```

Exemplo 3 - driverClientData.cpp (cont.)

```
   cout << left << setw( 10 ) << "Account" << setw(16)
        << "Last Name" << setw( 11 ) << "First Name"
        << left << setw( 10 ) << right << "Balance\n";

   ClientData client; // cria registro

   // lê o primeiro registro do arquivo
   inCredit.read(reinterpret_cast< char * >( &client ),
                 sizeof( ClientData ) );
    // lê todos os registros do arquivo
   while ( inCredit && !inCredit.eof() ) {
      // exibe o registro
      if ( client.getAccountNumber() != 0 )
         outputLine( cout, client );

      // lê o próximo registro do arquivo
      inCredit.read(reinterpret_cast<char*>(&client),
                    sizeof( ClientData ) );
   } // fim do while
   return 0;
} // fim de main
```

Exemplo 3 - driverClientData.cpp (cont.)

```
// exibe um único registro
void outputLine( ostream &output, const ClientData &
    record )
{
   output << left << setw( 10 )
          << record.getAccountNumber()
          << setw( 16 ) << record.getLastName()
          << setw( 11 ) << record.getFirstName()
          << setw( 10 ) << setprecision( 2 )
          << right << fixed << showpoint
          << record.getBalance() << endl;
} // fim da função outputLine
```

Exemplo 3 - driverClientData.cpp (cont.)

Account	Last Name	First Name	Balance
29	Brown	Nancy	−24.54
33	Dunn	Stacey	314.33
37	Barker	Doug	0.00
88	Smith	Dave	258.34
96	Stone	Sam	34.98

Exemplo 3 - driverClientData.cpp (cont.)

- Novamente, no método *read()* é necessário realizar a conversão do ponteiro para *const char*
- Enquanto não for o final do arquivo, determinado pelo método **eof()**, continuamos a ler do arquivo sequencialemente
  - Se a conta possuir número zero, significa que o registro está vazio, e portanto, não há dados

Exemplo 3 - driverClientData.cpp (cont.)

- A função *outputLine()* recebe como primeiro parâmetro uma referência de objeto da classe *ostream*
  - Pode ser um arquivo ou mesmo o *cout*
  - A mesma função pode ser utilizada para imprimir na saída padrão ou em um arquivo

Exemplo 3 - driverClientData.cpp (cont.)

- Note que, devido à forma em que o arquivo foi escrito, a leitura dos dados é realizada de forma ordenada em relação ao número da conta
  - A mesma ideia da escrita pode ser utilizada para ler um determinado registro

FIM