



BCC221

Programação Orientada a Objetos

Prof. Marco Antonio M. Carvalho

2014/2



UFOP

Endereços Importantes

- Site da disciplina:
<http://www.decom.ufop.br/marco/>
- Moodle:
www.decom.ufop.br/moodle
- Lista de e-mails:
bcc221-decom@googlegroups.com
- Para solicitar acesso:
<http://groups.google.com/group/bcc221-decom>



UFOP



Avisos



Avisos

UFOP



Na aula passada

UFOP

- Genéricos (*Templates*)
 - Programação Genérica
 - Funções Genéricas
 - Classes Genéricas
 - Outros Parâmetros e Parâmetros Padronizados
 - Observações Sobre Genéricos e Herança
 - Observações Sobre Genéricos e Funções Amigas
 - Observações Sobre Genéricos e Membros *static*
- *Standard Templates Library*
 - Contêineres
 - Iteradores
 - Algoritmos
 - Exemplos



Na aula de hoje

UFOP

- Processamento de Arquivos
 - Escrita
 - Leitura
 - Ponteiros de Posição
 - Arquivos de Acesso Aleatório
 - Exemplos



Arquivos e Persistência de Dados

UFOP

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



Arquivos

UFOP

- 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.
- Para realizarmos o processamento de arquivos, é necessário incluir os cabeçalhos **<iostream>** e **<fstream>**
 - *basic_ifstream*: Leitura de arquivos;
 - *basic_ofstream*: Escrita em arquivos;
 - *basic_fstream*: Leitura e escrita de arquivos.



Arquivos

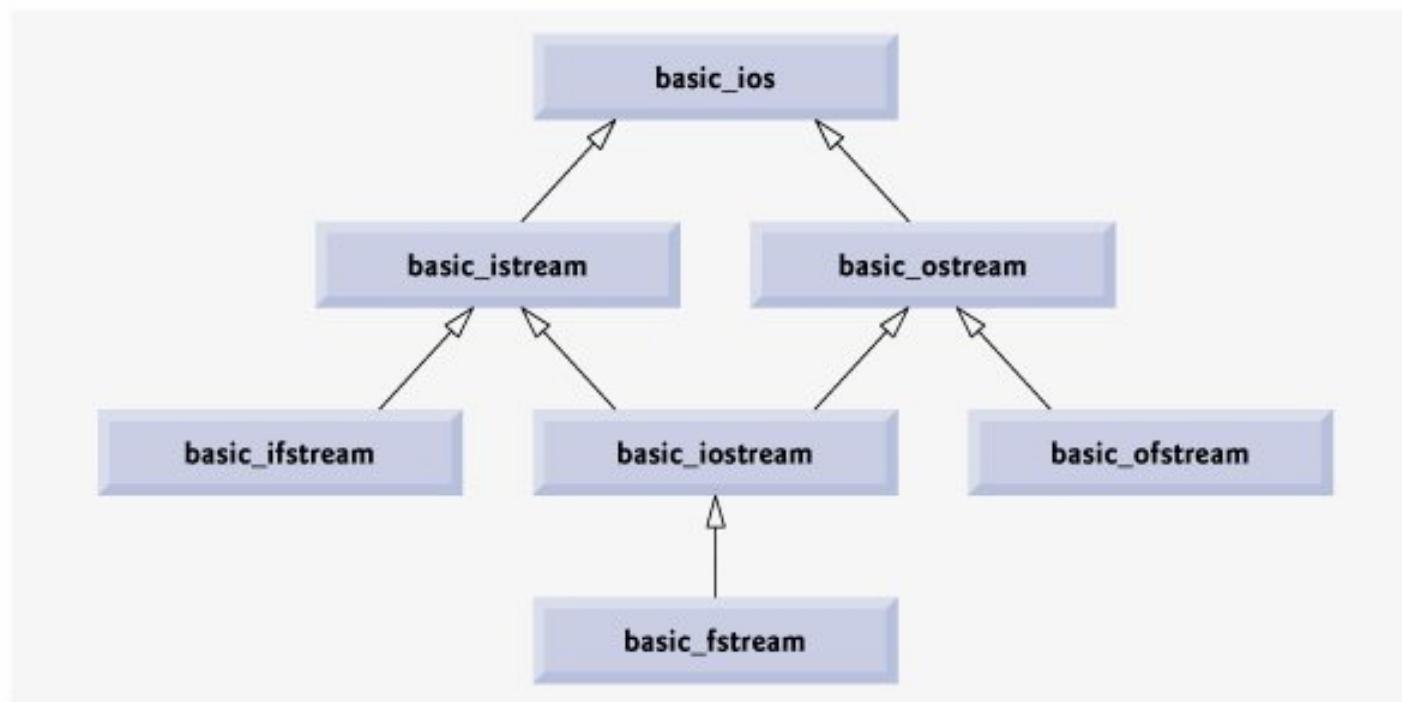
UFOP

- As classes listadas anteriormente são genéricas, e são pré-definidas para uma especialização do tipo *char*:
 - *ifstream*;
 - *ofstream*;
 - *fstream*.
- Um arquivo é manipulado através de objetos de uma destas especializações
 - Derivam das classes *basic_istream*, *basic_ostream* e *basic_iostream*;
 - Logo, todos os métodos, operadores e manipuladores também pertencem às classes genéricas e podem ser utilizados pelos arquivos.



Arquivos

UFOP



Escrita



Escrita

UFOP

- Vejamos um exemplo de 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.



Exemplo - Escrita

UFOP

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

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

    // fecha o programa se não conseguir criar arquivo
    if ( !outClientFile ) // operador ! sobre carregado
    {
        cerr << "O arquivo não pode ser aberto" << endl;
        exit( 1 );
    }
}
```



Exemplo - Escrita

UFOP

```
cout << "Informe a conta, o nome e o saldo." << endl
    << "Fim de arquivo para terminar a entrada.\n? ";

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 << "? ";
}

return 0; // destrutor ofstream fecha o arquivo
}
```



Saída

UFOP

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
```

fim-de-arquivo



UFOP

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



Escrita

UFOP

- No construtor, passamos dois argumentos
 - O nome do arquivo e o modo de abertura
 - *ios::out* para escrever os dados no arquivo, todo o conteúdo anterior é descartado;
 - Se o arquivo não existir, será criado;
 - *ios::app* para adicionar os dados ao conteúdo anterior do arquivo.



Arquivos

UFOP

Modo de Abertura	Descrição
<i>ios::app</i>	Adiciona os dados ao final do arquivo.
<i>ios::ate</i>	Abre o arquivo para escrita e se posiciona no final do arquivo. Dados podem ser escritos em qualquer posição do arquivo.
<i>ios::in</i>	Abre um arquivo para leitura.
<i>ios::out</i>	Abre um arquivo para escrita.
<i>ios::trunc</i>	Descarta o conteúdo de um arquivo, caso ele exista (esta é a ação padrão do <i>ios::out</i>).
<i>ios::binary</i>	Abre um arquivo para leitura ou escrita binária (não texto).



Arquivos

UFOP

- Note que 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

UFOP

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

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

Leitura



Leitura

UFOP

- Vejamos agora um exemplo de programa que lê os dados escritos no arquivo do exemplo anterior
 - Novamente, o construtor recebe o nome do arquivo e o modo de leitura como argumentos



Exemplo - Leitura

```
#include <iostream>
#include <fstream> // fluxo de arquivo
#include <iomanip>
#include <string>
#include <cstdlib>
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 << "O arquivo não pode ser aberto" << endl;
        exit( 1 );
    }
}
```



Exemplo - Leitura

UFOP

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

cout << left << setw( 10 ) << "Conta" << setw( 13 )
    << "Nome" << "Saldo" << 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

// 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;
}
```



Saída

UFOP

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



Leitura

UFOP

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

Ponteiros de Posição



Ponteiros de Posição

UFOP

- 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.



Ponteiros de Posição

UFOP

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



Ponteiros de Posição

UFOP

- O argumento para o método *seekg* normalmente é um *long*
 - Um segundo argumento pode ser especificado para indicar a direção
 - *ios::beg* – posicionamento relativo ao início do arquivo;
 - *ios::cur* – posicionamento relativo à posição atual;
 - *ios::end* – posicionamento relativo ao final do arquivo.
- As mesmas operações podem ser realizadas utilizando-se o método *seekp* da classe *ostream*.



Ponteiros de Posição

UFOP

```
// 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 );
```



Ponteiros de Posição

UFOP

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

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

Arquivos de Acesso Aleatório



Arquivos de Acesso Aleatório

UFOP

- Atualizar os dados de um arquivo sequencial não é tarefa fácil
 - Por exemplo, se temos um nome abreviado e queremos atualizá-lo para escrita por extenso, os próximos dados do arquivo podem ser corrompidos:

U. F. de Ouro Preto

Universidade Federal de Minas Gerais

Universidade Federa

l de Ouro Pretoderal de Minas Gerais



Arquivos de Acesso Aleatório

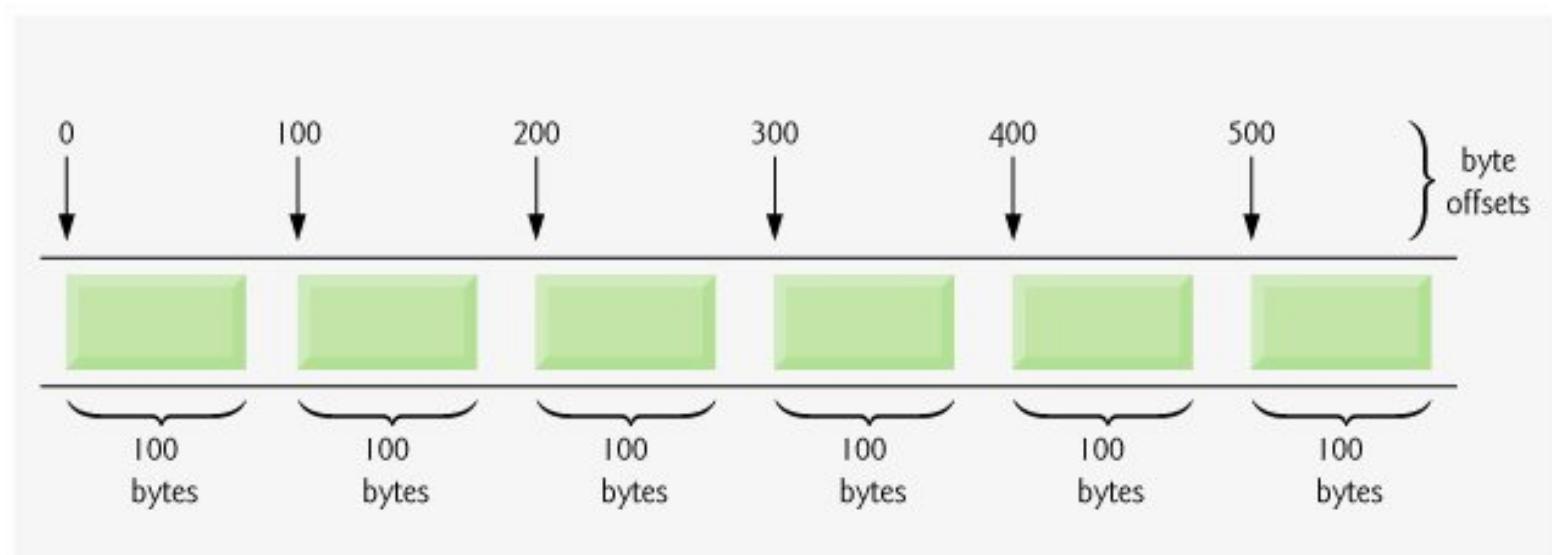
UFOP

- 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**
 - 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

UFOP





Arquivos de Acesso Aleatório

UFOP

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



Arquivos de Acesso Aleatório

UFOP

- O método ***write()*** da classe ***ostream*** escreve um número fixo de *bytes*, a partir de uma determinada posição
 - Quando o fluxo é uma arquivo, a posição é determinada pelo ponteiro de posição *put*.
- O método ***read()*** da classe ***istream*** lê um número fixo de *bytes*, a partir de uma determinada posição
 - Quando o fluxo é uma arquivo, a posição é determinada pelo ponteiro de posição *get*.



Arquivos de Acesso Aleatório

UFOP

- Ao escrevermos um número inteiro em um arquivo, ao invés de fazermos:

```
outfile << number;
```

- Podemos fazer:

```
outfile.write(reinterpret_cast< const char *>
(&number), sizeof(number));
```



Arquivos de Acesso Aleatório

UFOP

- A segunda opção sempre escreverá o inteiro como binário de *4 bytes*
 - O primeiro parâmetro é tratado como um grupo de *bytes*
 - Um ponteiro para *const char* possui apenas um *byte*.
 - A partir deste ponto, o método *write()* imprime a quantidade de bytes expressa pelo segundo argumento, um inteiro.
- Subsequentemente, o método *read()* pode ser utilizado para ler os quatro *bytes*.



Arquivos de Acesso Aleatório

UFOP

- 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 do ponteiro, não do valor apontado.

Exemplos



Exemplos

UFOP

- Nos exemplos a seguir, consideraremos 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

UFOP

- 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 exemplo escreve os dados para um arquivo e utiliza os métodos **seekp** e **write** para armazenar os dados em posições específicas do arquivo;
 - O terceiro exemplo lê os dados do arquivo de acesso aleatório e imprime somente os registros que contém dados.



ClientData.h

UFOP

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

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

```
// 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;
};
```



ClientData.cpp

UFOP

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

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

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



ClientData.cpp

UFOP

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

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

// 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
}
```



ClientData.cpp

UFOP

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

// 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
}
```



ClientData.cpp

UFOP

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

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

Exemplo 1



driverClientData.cpp

UFOP

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
#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 );
    }
```



driverClientData.cpp

UFOP

```
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;
}
```



Exemplo 1

UFOP

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

Exemplo 2



driverClientData.cpp

UFOP

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

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 );
    }
```



driverClientData.cpp

UFOP

```
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 );
```



driverClientData.cpp

```
// 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;
}

return 0;
}
```



Exemplo 2

UFOP

- 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 |.



Exemplo 2

UFOP

- Cada conta possui uma posição predeterminada no arquivo
 - A conta 1 é a primeira;
 - A conta 100 é a última.
- 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

UFOP

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>
using namespace std::exit;
#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 );
    }

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



driverClientData.cpp

UFOP

```
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 ) );
}

return 0;
}
```



driverClientData.cpp

UFOP

```
// 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;
}
```

Saída



UFOP

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

UFOP

- 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 sequencialmente
 - Se a conta possuir número zero, significa que o registro está vazio, e portanto, não há dados.



Exemplo 3

UFOP

- 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.
- 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 idéia da escrita pode ser utilizada para ler um determinado registro de acordo com o número da conta, pulando todos os demais com o ponteiro de posição.



UFOP



Perguntas?



Na próxima aula

- Prova



UFOP



FIM