



# Serialização e persistência de dados

Dr. Rodrigo Mologni Gonçalves dos Santos



# Sumário

1. Introdução
2. Arquivos e fluxos
3. Criando um arquivo sequencial
4. Lendo dados de um arquivo sequencial
5. Atualizando arquivos sequenciais
6. Arquivos de acesso aleatório
7. Criando um arquivo de acesso aleatório
8. Gravando dados aleatoriamente em um arquivo de acesso aleatório
9. Lendo um arquivo de acesso aleatório sequencialmente
10. Serialização de objetos

---


# Introdução

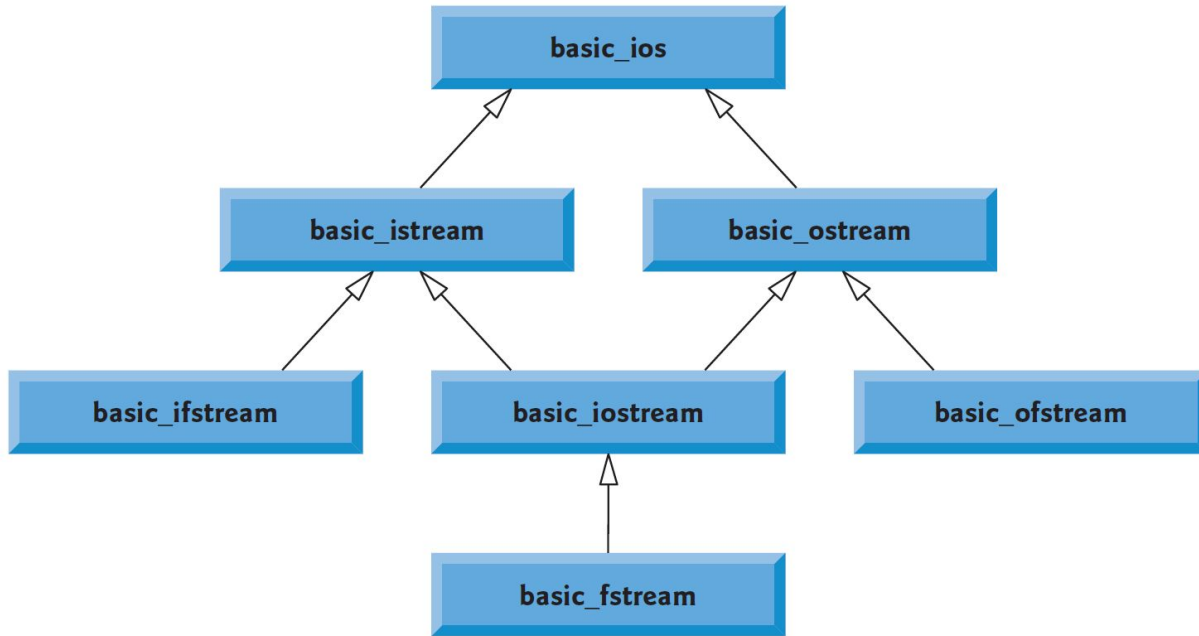


- Os arquivos são usados para **persistência de dados** (retenção permanente de dados).
- Os computadores armazenam arquivos em dispositivos de **armazenamento secundários**, como SSD e HD.

---

# Arquivos e fluxos

- 
- C++ visualiza cada arquivo como uma **sequência de *bytes***.
  - Cada arquivo termina com um **marcador de fim de arquivo** ou em um **número específico de *bytes*** registrado em uma estrutura de dados de um sistema de arquivos.
  - Quando um arquivo é aberto, um objeto é criado e um fluxo é associado ao objeto.



---

# Criando um arquivo sequencial





## Exemplo

- Abrir: Solução 'ch14' > **fig14\_03** > Arquivos de Origem > **Fig14\_03.cpp**



## Modo de abertura do fluxo

<b>app</b>	procura o final do fluxo antes de cada gravação
<b>binary</b>	abre em modo binário
<b>in</b>	abre para escrita
<b>out</b>	abre para leitura
<b>trunc</b>	descarta o conteúdo do fluxo ao abrir
<b>ate</b>	procura o final do fluxo imediatamente após abrir

---

# Lendo dados de um arquivo sequencial



## Exemplo

- Abrir: Solução 'ch14' > **fig14\_06** > Arquivos de Origem > **Fig14\_06.cpp**



## Ponteiros de posições de arquivo

```
ifs.seekg(n); // posição para o enésimo byte de ifs (supõe ios::beg)
ifs.seekg(n, ios::cur); // posiciona n bytes para frente em ifs
ifs.seekg(n, ios::end); // posiciona n bytes para trás a partir do fim de ifs
ifs.seekg(0, ios::end); // posiciona no fim de ifs
```



## Exemplo

- Abrir: Solução 'ch14' > **fig14\_07** > Arquivos de Origem > **Fig14\_07.cpp**

---

# Atualizando arquivos sequenciais



- Os **dados formatados** e gravados em um **arquivo sequencial** não podem ser modificados sem o risco de destruir outros dados do arquivo. O problema é que os registros podem variar em tamanho.





## Expectativa

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

```
100 Jones 24.98
200 Doe 345.67
300 Worthington 0.00
400 Stone -42.16
500 Rich 224.62
```




## Realidade


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

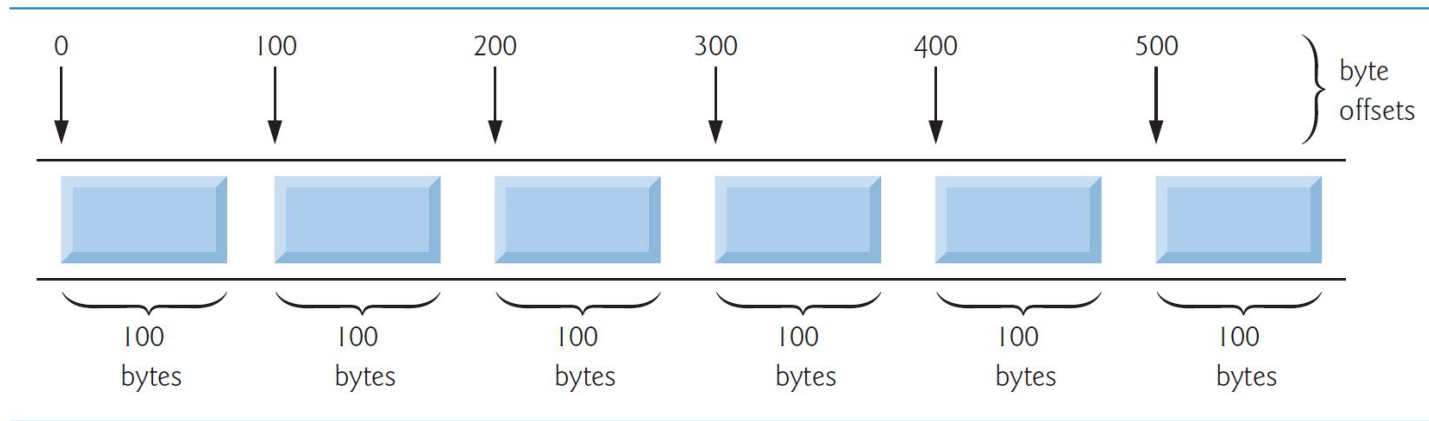
```
100 Jones 24.98
200 Doe 345.67
300 Worthington 400 Stone -42.16
500 Rich 224.62
```

---

# Criando um arquivo de acesso aleatório

- 
- Os arquivos sequenciais são inadequados aos aplicativos de **acesso instantâneo** (como sistemas de processamento de transação), em que um registro particular deve ser imediatamente localizado.
  - Esse tipo de acesso instantâneo é possível com **arquivos de acesso aleatório**. Os registros individuais de um arquivo de acesso aleatório podem ser acessados diretamente (e rapidamente) sem precisar pesquisar outros registros.

- 
- O método mais fácil seja impor que todos os registros em um arquivo tenham o mesmo comprimento fixo. A utilização de registros de mesmo tamanho e largura fixa facilita para um programa calcular (como uma função do tamanho do registro e da chave do registro) a localização exata de qualquer registro em relação ao início do arquivo.
  - Os dados podem ser inseridos em um arquivo de acesso aleatório sem destruir outros dados no arquivo.
  - Os dados previamente armazenados também podem ser atualizados ou excluídos sem regravar o arquivo inteiro.





## Exemplo

- Abrir: Solução 'ch14' > **fig14\_09\_11** > Arquivos de Origem > **Fig14\_11.cpp**

---

# Gravando dados aleatoriamente em um arquivo de acesso aleatório





## Exemplo

- Abrir: Solução 'ch14' > **fig14\_12** > Arquivos de Origem > **Fig14\_12.cpp**



```
basic_ostream& write( const char_type* s, std::streamsize count );  
reinterpret_cast< target-type >( expression );
```

---

# Lendo um arquivo de acesso aleatório sequencialmente



## Exemplo

- Abrir: Solução 'ch14' > **fig14\_13** > Arquivos de Origem > **Fig14\_13.cpp**

---

# Serialização de objetos



## Problema

- Quando os membros dos dados do objeto são enviados para um arquivo, **perde-se** as informações do **tipo do objeto**. Apenas os **valores dos atributos do objeto**, e **não** as **informações de tipo**, são armazenadas. Se o programa que lê esses dados **souber** o **tipo de objeto** ao qual os dados correspondem, o programa poderá ler os dados de um objeto desse tipo.



## Solução

- Um **objeto serializado** é um objeto representado como uma sequência de *bytes* que inclui os **dados do objeto**, bem como informações sobre o **tipo do objeto** e os **tipos de dados armazenados no objeto**. Um **objeto serializado** pode ser lido do arquivo e **desserializado**.
- As bibliotecas [Boost](#), de código-aberto, fornecem suporte para serialização de objetos em formatos de texto, binários e linguagem de marcação extensível (XML).

---

# Referências





## Chapter 14: File Processing

