



2º Trabalho¹

Playlist

Segunda-feira, 15 de março de 2021.

Desenvolva um aplicativo chamado Playlist, lista de músicas, que possui as seguintes classes.

1. Musica: possui três atributos, número de músicas, título e artista, dois construtores e os métodos de acesso aos atributos. O atributo número de músicas deve ser uma variável de classe.
2. ArquivoMusica: implementa os métodos abaixo de modo a possibilitar a escrita e leitura de um objeto da classe Musica em um arquivo de acesso aleatório ou direto.

```
// Cria um objeto para manipular o arquivo binário com acesso aleatório.
```

```
ArquivoMusica();
```

```
/* Cria um objeto para manipular o arquivo binário com acesso aleatório cujo nome de arquivo está  
especificado em nomeArquivo. Abre o arquivo para leitura e escrita.
```

```
*/
```

```
ArquivoMusica(string nomeArquivo);
```

```
// Exclui o objeto arquivo binário.
```

```
~ArquivoMusica();
```

```
/* Abre o arquivo com o nome especificado em nomeArquivo para escrita e leitura de dados.  
Retorna true se o arquivo foi aberto com sucesso e false caso contrário.
```

```
*/
```

```
bool abrir(string nomeArquivo);
```

```
// Fecha o arquivo.
```

```
void fechar();
```

```
// Obtém o nome do arquivo.
```

```
string getNomeArquivo();
```

```
// Obtém o número de registros do arquivo.
```

```
unsigned int numeroRegistros();
```

```
// Obtém o tamanho do registro em bytes.
```

```
unsigned int tamanhoRegistro();
```

```
// Escreve o objeto Musica como um registro do arquivo.
```

```
void escrever(Musica musica);
```

¹ Atualizado em 16/03/2021.

```

/* Lê os dados de um registro do arquivo e armazena-os no objeto Musica.
   Retorna o objeto Musica, e em caso de erro nullptr.
*/
Musica* ler(unsigned int numeroRegistro);

/* Pesquisa o título de uma música no arquivo. Em caso de sucesso retorna o número do registro
   onde a música está armazenada e -1 caso contrário.
*/
int pesquisarTituloMusica(string tituloMusica);

/* Pesquisa o artista de uma música no arquivo. Em caso de sucesso retorna o número do registro
   onde a música está armazenada e -1 caso contrário.
*/
int pesquisarArtista(string artista);

```

3. Índice: o objetivo dessa classe é servir de base para construir um arquivo de índice que permita localizar um registro em um arquivo de acesso aleatório de acordo com o seu campo de identificação única (chave primária). A classe Índice possui dois atributos, chavePrimaria e numeroRegistro, um construtor e métodos de acesso. O atributo chavePrimaria representa o valor de um campo usado como chave primária para acessar um registro do arquivo, e o atributo numeroRegistro é o número do registro do arquivo que possui na sua chave primária um valor igual ao atributo chavePrimaria.
4. ArquivoIndice: essa classe deve implementar os mesmos métodos da classe ArquivoMusica para que a escrita e leitura de um objeto da classe Índice sejam realizadas em um arquivo de acesso aleatório.

As funcionalidades abaixo devem ser criadas em uma classe chamada Playlist.

1. Criar *Playlist*

Permite que o usuário forneça um nome de diretório que possui uma coleção de músicas em formato MP3. Todos os arquivos dessa pasta devem ser lidos e armazenados em um arquivo de acesso aleatório chamado Playlist.dat. O artista e o título da música devem ser extraídos do nome do arquivo MP3, que possui o formato abaixo.

Artista - TituloMúsica.mp3

Exemplos:

Snow Patrol - Just Say Yes.mp3
Legião Urbana - Pais e Filhos.mp3

Cada registro do arquivo Playlist.dat deve ser composto de dois campos, artista e tituloMusica. Portanto, Playlist.dat é um arquivo usado para armazenar objetos da classe Musica usando os serviços da classe ArquivoMusica. A extensão MP3 deve ser removida do nome do arquivo para não ser incluída no cadastro da música no arquivo Playlist.dat.

Veja na tabela abaixo um exemplo desse arquivo.

Registro	artista	tituloMusica
0	Kiko Zambianchi	Primeiros Erros
1	Cláudio Zoli	Noite do Prazer
2	Gilberto Gil	Vamos Fugir
3	Barão Vermelho	Por Você

Tabela 1 - Exemplo de arquivo Playlist.dat

Após o programa lê os arquivos de música do diretório, extrair o artista e o título de cada música para criar o arquivo Playlist.dat, ele deve gerar dois arquivos de índices de acesso aleatório para indexar o arquivo Playlist.dat pelo título da música (ArqIndTituloMusica.dat) e pelo artista (ArqIndArtista.dat).

- ArqIndTituloMusica: cada registro deve ser composto de dois campos, chavePrimaria e numeroRegistro, onde o primeiro armazena o título da música e o segundo o número do registro do arquivo Playlist.dat onde a música está armazenada. Observe na tabela abaixo que o arquivo está classificado pelo título da música, ou seja, esse é um índice da *playlist* ordenado pelo título das músicas dessa coleção.

Registro	chavePrimaria	numeroRegistro
0	Noite do Prazer	1
1	Por Você	3
2	Primeiros Erros	0
3	Vamos Fugir	2

Tabela 2 - Exemplo de arquivo de índice para o arquivo Playlist.dat da Tabela 1.

- ArqIndArtista: cada registro deve ser composto de dois campos, chavePrimaria e numeroRegistro, onde o primeiro armazena o artista e o segundo o número do registro do arquivo Playlist.dat onde o artista está armazenado. Observe na tabela abaixo que o arquivo está classificado pelo nome do artista, ou seja, esse é um índice da *playlist* ordenado pelo artista das músicas dessa coleção.

Registro	chavePrimaria	numeroRegistro
0	Barão Vermelho	3
1	Cláudio Zoli	1
2	Gilberto Gil	2
3	Kiko Zambianchi	0

Tabela 3 - Exemplo de arquivo de índice para o arquivo Playlist.dat da Tabela 1.

Esses dois arquivos de índice são criados para armazenar objetos da classe Indice usando os serviços da classe ArquivoIndice, e devem ter os seus registros ordenados ascendentemente pelo valor da chave primária.

Após a criação da *playlist* e geração dos arquivos de índices o programa deve exibir a mensagem abaixo.

Playlist criada com 2.500 músicas.
Arquivos de índice criados.

A variável de classe do tipo de dado Musica deve ser usado para contabilizar o número de músicas localizadas no diretório informado pelo usuário, uma vez que pra cada música dessa pasta deve-se criar um objeto Musica para ser gravado no arquivo Playlist.dat. Ao exibir a mensagem acima essa variável de classe deve ser usada para informar o número de músicas da *playlist*.

2. Exibir *Playlist* classificada pelo título da música

Exibe um relatório com todas as músicas do arquivo Playlist.dat ordenadas ascendentemente pelo título da música. Mostra primeiro o título da música seguido pelo nome do artista, ambos separados por um espaço, um hífen e um espaço (" - "). Esses dados são lidos sequencialmente do arquivo Playlist.dat para a memória, classificados e exibidos usando o leiaute abaixo. Observe que os títulos das músicas devem ser numerados.

Playlist classificada pelo título da música
Número de Músicas = 7

1. Bad - Michael Jackson
2. Beat It - Michael Jackson
3. Heal The World - Michael Jackson
4. Rolling In The Deep - Adele
5. Someone Like You - Adele
6. Still Loving You - Scorpions
7. Thriller - Michael Jackson

3. Exibir *Playlist* classificada pelo nome do artista

Exibe um relatório com todas as músicas do arquivo Playlist.dat ordenadas ascendentemente pelo nome do artista. Mostra primeiro o artista seguido pelo título da música, ambos separados por um espaço, um hífen e um espaço (" - "). Esses dados são lidos sequencialmente do arquivo Playlist.dat para a memória, classificados e exibidos usando o leiaute abaixo. Observe que os nomes dos artistas devem ser numerados.

O relatório deve ter o leiaute abaixo.

Playlist classificada pelo artista
Número de Músicas = 7


1. Adele - Rolling In The Deep
2. Adele - Someone Like You
3. Michael Jackson - Bad
4. Michael Jackson - Beat It
5. Michael Jackson - Heal The World
6. Michael Jackson - Thriller
7. Scorpions - Still Loving You

4. Pesquisar música

O usuário deve fornecer o título da música e o programa deve pesquisá-lo no arquivo de índice ArqIndTituloMusica.dat. Se o título for encontrado, o programa deve exibir primeiro o título da música seguido pelo nome do artista, ambos separados por um espaço, um hífen e um espaço (" - "). Esses dados são obtidos do arquivo Playlist.dat via acesso direto ao registro, cujo número foi obtido após consulta no arquivo de índice. Em caso de falha na pesquisa deve-se exibir a mensagem Música não encontrada.

5. Pesquisar artista

O usuário deve fornecer o nome do artista e o programa deve pesquisá-lo no arquivo de índice `ArqIndArtista.dat`. Se o artista for encontrado, o programa deve exibir primeiro o artista seguido pelo título da música, ambos separados por um espaço, um hífen e um espaço (" - "). Esses dados são obtidos do arquivo `Playlist.dat` via acesso direto ao registro, cujo número foi obtido após consulta no arquivo de índice. Em caso de falha na pesquisa deve-se exibir a mensagem `Artista não encontrado`.

 **Nota:** Observe que, nas funções 4 e 5 do programa, os arquivos de índice são utilizados para pesquisar o título da música ou o nome do artista para obter o número do registro em que a música está localizada no arquivo `Playlist.dat`. Isso é feito para que a funcionalidade essencial do arquivo de índice seja empregada para localizar rapidamente esse registro na coleção de músicas, ou seja, no arquivo `Playlist.dat`. Para pesquisar o título da música e o artista nos arquivos de índice deve-se usar o algoritmo de pesquisa binária por meio da função `binary_search`, definida no arquivo de cabeçalho `algorithm` da *C++ Standard Library*. Localizado o número do registro, basta fazer um acesso direto por meio de uma função de busca que movimenta o ponteiro do arquivo `Playlist.dat` diretamente para o registro (*seek*).

- Critérios de avaliação

1. O trabalho será avaliado considerando:
 - a. A validação dos dados fornecidos pelo usuário.
 - b. A lógica empregada na solução do problema.
 - c. O funcionamento do programa.
 - d. O conhecimento da linguagem de programação C++.
 - e. A implementação dos conceitos de orientação a objetos.
 - f. O uso do princípio do menor privilégio².
 - g. Código fonte sem erros e sem advertências do compilador.
 - h. Código fonte legível, indentado, organizado e comentado.
 - i. Identificadores significativos para aprimorar a inteligibilidade do código fonte.
2. O programa deve ser desenvolvido integralmente usando apenas os recursos da linguagem C++ e do *Microsoft Visual Studio Community* 2019, versão 16.8. Programas desenvolvidos em outras linguagens, mesmo que parcialmente, receberão nota zero.
3. Para que o programa seja avaliado o código deve executar com sucesso. Programas que apresentarem erros de compilação e/ou ligação receberão nota zero.
4. Trabalhos com plágio, ou seja, programas com código fonte copiados de outra pessoa (cópia integral ou parcial) receberão nota zero.
5. O desenvolvimento do trabalho é individual.
6. Incluir em cada classe apenas as definições de tipos de dados, variáveis, constantes e métodos que forem essenciais para a funcionalidade da mesma.

² O **princípio do menor privilégio** declara que deve ser concedido ao código somente a quantidade de privilégio e acesso de que ele precisa para realizar sua tarefa designada, não mais que isso.

7. Escrever funções e métodos específicos, ou seja, com atribuição clara e objetiva.

Exemplo: Pesquisa um nome em um vetor de *strings*. Retorna a posição do nome no vetor se ele for encontrado ou -1 caso contrário.

```
int pesquisarNome(const string vetor[], string nome);
```

A descrição dessa função deixa claro que não é atribuição dela ler o nome via algum dispositivo de E/S e nem exibir o resultado da consulta, somente realizar a pesquisa do nome no vetor e devolver o resultado.

O uso do *const* no protótipo de função acima é um exemplo do princípio do menor privilégio, porque se a função não precisa alterar os argumentos usados na sua chamada, ela não pode ter parâmetros formais com esse poder ou privilégio. O uso do *const* assegura que apesar da função receber um vetor, que sempre é passado por referência, ela não poderá modificá-lo.

8. Não escrever código redundante.

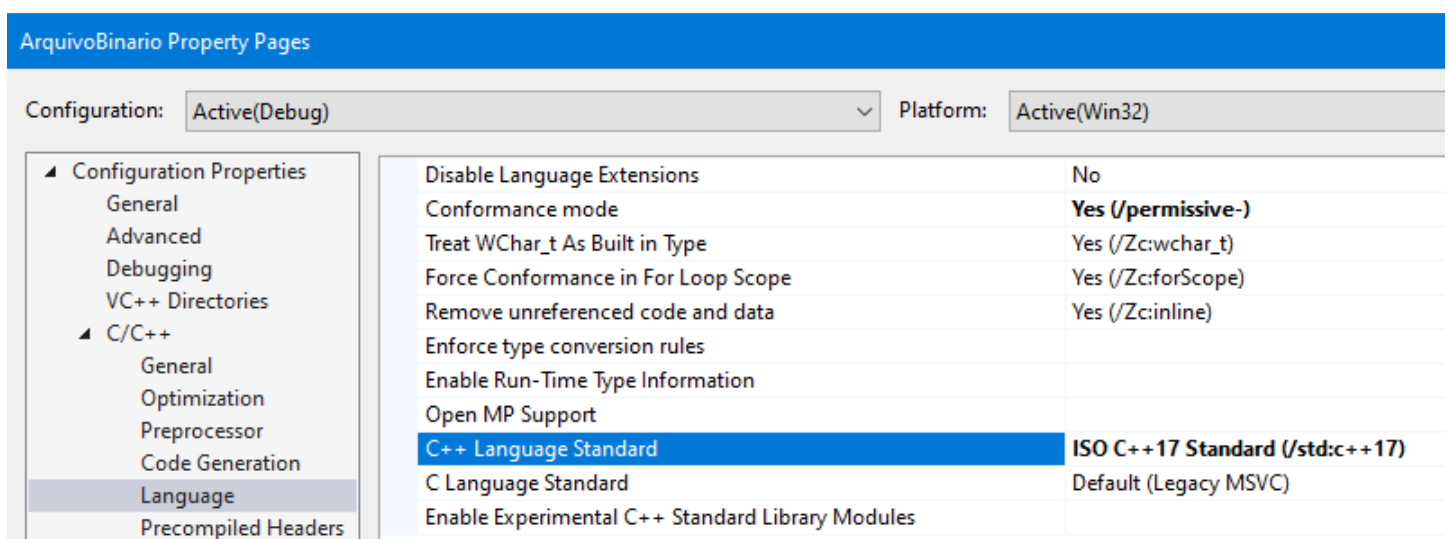
9. É proibido modificar os nomes de arquivos, identificadores, os protótipos de função, as declarações e/ou definições de métodos e classes fornecidos neste texto ou em anexo.

10. É permitido acrescentar novas declarações e/ou definições de classes, métodos, variáveis e constantes desde que estejam de acordo com os critérios acima.

11. Use a classe *ArquivoBinario* fornecida no projeto *ArquivoBinario.7z* para criar os arquivos binários de acesso aleatório. Use a interface e implementação da classe *ArquivoProduto* para criar as interfaces e implementações para as classes *ArquivoMusica* e *ArquivoIndice*.

12. A classe *Playlist* deve usar ponteiros inteligentes (*smart pointers*) para armazenar em um vector os objetos da classe *Musica*.

13. Para acessar os arquivos MP3 no sistema de arquivos do sistema operacional deve-se usar a *Filesystem library*³ introduzida no C++17. Portanto, configure a versão da Linguagem C++ usada no seu projeto por meio da janela *Property Pages* (*menu* Project, opção *Properties*) com a opção *ISO C++17 Standard*, veja um exemplo abaixo.



³ *Filesystem library*: <https://en.cppreference.com/w/cpp/filesystem>

- Instruções para entrega do trabalho

1. Crie uma solução com o nome Playlist e um projeto com o seu nome e sobrenome, por exemplo: AyrtonSenna.
2. Limpe a solução para apagar todos os arquivos OBJ da pasta *Debug* do projeto. Confira se esses arquivos realmente foram excluídos da pasta *Debug*. De preferência exclua todo o conteúdo dessa pasta.
3. Antes de submeter os exercícios via SIGAA, compacte apenas o diretório do projeto para criar um arquivo 7z com o seu nome e sobrenome, por exemplo: AyrtonSenna.7z.

Não inclua no arquivo 7z o diretório da solução Playlist, somente o diretório do projeto que possui o seu nome e sobrenome.

Assim o tamanho final do arquivo 7z não ultrapassará o limite de 10 MB do SIGAA, porque o conteúdo do diretório oculto *.vs*, criado dentro da pasta da solução, não será compactado, reduzindo drasticamente o tamanho final do arquivo.

Para compactar o projeto use o *software* livre de código aberto 7-Zip, que está disponível em <https://www.7-zip.org/download.html>.

- Data de entrega

Segunda-feira, 22 de março de 2021.

- Valor do trabalho

10,0 pontos.

Prof. Márlon Oliveira da Silva
marlon.silva@ifsudestemg.edu.br