Mp3

1.0v

Gerado por Doxygen 1.9.7

Chapter 1

README

College Project Part. 2

Project making a Mp3 using C++ at the college subject ¨Programing Language¨.

Como compilar:

- Na pasta raiz do projeto, utilize os comandos a seguir:

```
cmake -B build
cd build
make
```

- Utilize o comando a seguir para rodar:
  - foi feito um arquivo de teste ¨musicas.txt¨.

```
./mp3 ../musicas.txt
```

Colaborador: `@joapedu`
João Eduardo - 20220035851

:phone: *C O N T A T O S* :phone:

Colaborador: `@edurs2602`
Luis Eduardo - 20220028973

:phone: *C O N T A T O S* :phone:

Chapter 2

Índice dos Componentes

2.1 Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

ListaEncadeada< T > . . . . . **??**

Musica

Classe que representa uma música . . . . . **??**

No< T > . . . . . **??**

Playlist

Classe que implementa uma playlist . . . . . **??**

Chapter 3

Índice dos Arquivos

3.1 Lista de Arquivos

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

include/listaencadeada.hpp . . . . . ??
include/menu.hpp . . . . . ??
include/musica.hpp . . . . . ??
include/no.hpp . . . . . ??
include/playlist.hpp . . . . . ??
src/arquivo.cpp . . . . . ??

Chapter 4

Classes

4.1 Referência do Template da Classe ListaEncadeada< T >

`#include <listaencadeada.hpp>`

Membros Públicos

- ListaEncadeada ()
- ListaEncadeada (const ListaEncadeada< T > &outra)
- ~ListaEncadeada ()
- void limpar ()
- size_t getTamanho () const
- No< T > * getCabeca () const
- No< T > * getCauda () const
- void setCabeca (No< T > *cabeca)
- void setCauda (No< T > *cauda)
- void adicionar (T valor)
- void adicionar (const ListaEncadeada< T > &outra)
- T * buscarValor (T valor)
- void removerFinal ()
- int removerValor (T valor)
- void removerValor (ListaEncadeada< T > &outra)
- void imprimir ()
- ListaEncadeada< T > operator+ (ListaEncadeada< T > &outra)
- const ListaEncadeada< T > & operator>> (No< T > &no)
- const ListaEncadeada< T > operator<< (const No< T > &no)

Atributos Privados

- No< T > * **cabeca**

  Ponteiro para o primeiro elemento da lista.
- No< T > * **cauda**

  Ponteiro para o último elemento da lista.

4.1.1 Descrição detalhada

template<typename T>
class ListaEncadeada< T >

Classe que implementa uma lista encadeada genérica.

Parâmetros do template

T	O tipo de dado que a lista irá conter.

4.1.2 Construtores e Destrutores

4.1.2.1 ListaEncadeada() [1/2]

```
template<typename T >
ListaEncadeada< T >::ListaEncadeada
```

Construtor da lista encadeada.

4.1.2.2 ListaEncadeada() [2/2]

```
template<typename T >
ListaEncadeada< T >::ListaEncadeada (
            const ListaEncadeada< T > & outra )
```

Construtor cópia da lista encadeada.

Parâmetros

outra	A lista encadeada a ser copiada.

4.1.2.3 ~ListaEncadeada()

```
template<typename T >
ListaEncadeada< T >::~ListaEncadeada
```

Destrutor da lista encadeada, que remove todos os elementos.

4.1.3 Documentação das funções

4.1.3.1 adicionar() [1/2]

```
template<typename T >
void ListaEncadeada< T >::adicionar (
            const ListaEncadeada< T > & outra )
```

Adiciona todos os elementos de uma lista ao final da lista.

Parâmetros

outra	A lista contendo os elementos a serem adicionados.

4.1.3.2 adicionar() [2/2]

```
template<typename T >
void ListaEncadeada< T >::adicionar (
            T valor )
```

Adiciona um novo elemento com valor especificado ao final da lista.

Parâmetros

valor	O valor a ser adicionado à lista.

4.1.3.3 buscarValor()

```
template<typename T >
T * ListaEncadeada< T >::buscarValor (
            T valor )
```

Procura um elemento específico na lista.

Parâmetros

valor	O valor do elemento a ser procurado.

Retorna

Um ponteiro para o valor encontrado ou nullptr se não encontrado.

4.1.3.4 getCabeca()

```
template<typename T >
No< T > * ListaEncadeada< T >::getCabeca
```

Retorna o início da lista.

Retorna

O primeiro elemento da lista.

4.1.3.5 getCauda()

```
template<typename T >
No< T > * ListaEncadeada< T >::getCauda
```

Retorna o final da lista.

Retorna

O último elemento da lista.

4.1.3.6 getTamanho()

```
template<typename T >
size_t ListaEncadeada< T >::getTamanho
```

Retorna o tamanho da lista encadeada.

Retorna

O tamanho da lista.

4.1.3.7 imprimir()

```
template<typename T >
void ListaEncadeada< T >::imprimir
```

Imprime todos os elementos da lista recursivamente.

4.1.3.8 limpar()

```
template<typename T >
void ListaEncadeada< T >::limpar
```

Remove todos os elementos da lista.

4.1.3.9 operator+()

```
template<typename T >
ListaEncadeada< T > ListaEncadeada< T >::operator+ (
            ListaEncadeada< T > & outra )
```

Sobrecarga do operador de soma, que concatena duas listas.

Parâmetros

outra	A lista a ser concatenada com esta.

Retorna

Uma nova lista contendo os elementos das duas listas concatenados.

4.1.3.10 operator<<()

```
template<typename T >
const ListaEncadeada< T > ListaEncadeada< T >::operator<< (
            const No< T > & no )
```

Sobrecarga do operador de inserção, que adiciona um elemento à lista.

Parâmetros

no	O nó contendo o valor a ser adicionado à lista.

Retorna

A lista após a inserção.

4.1.3.11 operator>>()

```
template<typename T >
const ListaEncadeada< T > & ListaEncadeada< T >::operator>> (
            No< T > & no )
```

Sobrecarga do operador de extração, que remove um elemento da lista.

Parâmetros

no	Referência ao nó onde o valor removido será armazenado.

Retorna

A lista após a extração.

4.1.3.12 removerFinal()

```
template<typename T >
void ListaEncadeada< T >::removerFinal
```

Remove o elemento no final da lista.

4.1.3.13 removerValor() [1/2]

```
template<typename T >
void ListaEncadeada< T >::removerValor (
            ListaEncadeada< T > & outra )
```

Remove todos os elementos de uma lista da lista.

Parâmetros

outra	A lista contendo os elementos a serem removidos.

4.1.3.14 removerValor() [2/2]

```
template<typename T >
```

```
int ListaEncadeada< T >::removerValor (
            T valor )
```

Remove o elemento especificado da lista.

Parâmetros

valor	O valor do elemento a ser removido.

Retorna

1 se o elemento foi removido com sucesso, 0 se não encontrado.

4.1.3.15 setCabeca()

```
template<typename T >
void ListaEncadeada< T >::setCabeca (
            No< T > * cabeca )
```

Altera o ponteiro cabeça da lista.

Parâmetros

cabeca	O novo valor para o ponteiro cabeça.

4.1.3.16 setCauda()

```
template<typename T >
void ListaEncadeada< T >::setCauda (
            No< T > * cauda )
```

Altera o ponteiro cauda da lista.

Parâmetros

cauda	O novo valor para o ponteiro cauda.

A documentação para essa classe foi gerada a partir do seguinte arquivo:

- include/listaencadeada.hpp

4.2 Referência da Classe Musica

Classe que representa uma música.

```
#include <musica.hpp>
```

Membros Públicos

- Musica ()
  Construtor padrão da classe Musica.
- Musica (std::string titulo, std::string autor="")
  Construtor da classe Musica que recebe título e autor.
- std::string getTitulo ()
  Obtém o título da música.
- std::string getAutor ()
  Obtém o autor da música.
- void setTitulo (std::string titulo)
  Altera o título da música.
- void setAutor (std::string autor)
  Altera o autor da música.
- bool operator== (Musica &outra)
  Sobrecarga do operador de igualdade.

Atributos Privados

- std::string **titulo**
  Título da música.
- std::string **autor**
  Autor da música.

Amigos

- std::ostream & operator<< (std::ostream &os, const Musica &musica)
  Sobrecarga do operador de inserção para imprimir informações da música.

4.2.1 Descrição detalhada

Classe que representa uma música.

4.2.2 Construtores e Destrutores

4.2.2.1 Musica() [1/2]

```
Musica::Musica ( )
```

Construtor padrão da classe Musica.

Cria um objeto Musica com título e autor vazios.

4.2.2.2 Musica() [2/2]

```
Musica::Musica (
            std::string titulo,
            std::string autor = "" )
```

Construtor da classe Musica que recebe título e autor.

Parâmetros

titulo	O título da música.
autor	O autor da música. (opcional, padrão = "")

Parâmetros

titulo	Título da música.
autor	Autor da música.

4.2.3 Documentação das funções

4.2.3.1 getAutor()

```
std::string Musica::getAutor ( )
```

Obtém o autor da música.

Retorna

O autor da música como uma string.

Retorna

O autor da música como uma string.

4.2.3.2 getTitulo()

```
std::string Musica::getTitulo ( )
```

Obtém o título da música.

Retorna

O título da música como uma string.

Retorna

O título da música como uma string.

4.2.3.3 operator==()

```
bool Musica::operator== (
            Musica & outra )
```

Sobrecarga do operador de igualdade.

Sobrecarga do operador de igualdade para a classe Musica.

Parâmetros

outra	A outra música para comparar.

Retorna

True se as músicas forem iguais, False caso contrário.

Verifica se duas músicas são iguais comparando seus títulos.

Parâmetros

outra	A música a ser comparada.

Retorna

true se as músicas têm o mesmo título, false caso contrário.

4.2.3.4 setAutor()

```
void Musica::setAutor (
            std::string autor )
```

Altera o autor da música.

Define o autor da música.

Parâmetros

autor	O novo autor da música.

Parâmetros

autor	O autor da música a ser definido.

4.2.3.5 setTitulo()

```
void Musica::setTitulo (
            std::string titulo )
```

Altera o título da música.

Define o título da música.

Parâmetros

titulo	O novo título da música.

Parâmetros

titulo	O título da música a ser definido.

4.2.4 Documentação dos símbolos amigos e relacionados

4.2.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Musica & musica )  [friend]
```

Sobrecarga do operador de inserção para imprimir informações da música.

Parâmetros

os	O stream de saída onde a música será impressa.
musica	A música a ser impressa.

Retorna

O stream de saída após a impressão.

Parâmetros

os	O stream de saída.
musica	A música a ser exibida no stream.

Retorna

O stream de saída com as informações da música formatadas.

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- include/musica.hpp
- src/musica.cpp

4.3 Referência do Template da Classe No< T >

```
#include <no.hpp>
```

Membros Públicos

- No ()=default
- No (const T &valor)
- T & getValor ()
- No * getProximo ()
- void setValor (T valor)
- void setProximo (No< T > *proximo)

Atributos Privados

- T **valor**

  O valor armazenado no nó.
- No< T > * **proximo**

  Ponteiro para o próximo nó na lista.

4.3.1 Descrição detalhada

template<typename T>
class No< T >

Classe do nó da lista encadeada.

Parâmetros do template

T	O tipo de dado que o nó irá armazenar.

4.3.2 Construtores e Destrutores

4.3.2.1 No() [1/2]

```
template<typename T >
No< T >::No ( )  [default]
```

Construtor padrão do nó.

4.3.2.2 No() [2/2]

```
template<typename T >
No< T >::No (
            const T & valor )
```

Construtor que recebe o valor a ser colocado no nó.

Parâmetros

valor	O valor a ser armazenado no nó.

Construtor que recebe o valor a ser colocado no nó.

Parâmetros do template

T	O tipo de dado que o nó irá armazenar.

Parâmetros

valor	O valor a ser armazenado no nó.

4.3.3 Documentação das funções

4.3.3.1 getProximo()

```
template<typename T >
No< T > * No< T >::getProximo
```

Obtém o ponteiro para o próximo nó.

Retorna

O ponteiro para o próximo nó na lista.

Obtém o ponteiro para o próximo nó.

Parâmetros do template

T	O tipo de dado que o nó armazena.

Retorna

O ponteiro para o próximo nó na lista.

4.3.3.2 getValor()

```
template<typename T >
T & No< T >::getValor
```

Obtém o valor do nó atual.

Retorna

Uma referência ao valor armazenado no nó.

Obtém o valor do nó atual.

Parâmetros do template

T	O tipo de dado que o nó armazena.

Retorna

Uma referência ao valor armazenado no nó.

4.3.3.3 setProximo()

```
template<typename T >
void No< T >::setProximo (
            No< T > * proximo )
```

Altera o ponteiro para o próximo nó.

Parâmetros

proximo	O novo ponteiro para o próximo nó na lista.

Altera o ponteiro para o próximo nó.

Parâmetros do template

T	O tipo de dado que o nó armazena.

Parâmetros

proximo	O novo ponteiro para o próximo nó na lista.

4.3.3.4 setValor()

```
template<typename T >
void No< T >::setValor (
            T valor )
```

Altera o valor do nó atual.

Parâmetros

valor	O novo valor a ser armazenado no nó.

Altera o valor do nó atual.

Parâmetros do template

T	O tipo de dado que o nó armazena.

Parâmetros

valor	O novo valor a ser armazenado no nó.

A documentação para essa classe foi gerada a partir do seguinte arquivo:

- include/no.hpp

4.4 Referência da Classe Playlist

Classe que implementa uma playlist.

`#include <playlist.hpp>`

Membros Públicos

- Playlist ()
  Construtor padrão da classe Playlist.
- Playlist (std::string nome)
  Construtor da playlist que recebe seu nome.
- Playlist (const Playlist &outra)
  Construtor cópia da playlist.
- ~Playlist ()
  Destrutor da playlist, que remove todas as músicas.
- size_t getTamanho ()
  Obtém o tamanho da playlist (quantidade de músicas).
- const std::string & getNome () const
  Obtém o nome da playlist.
- ListaEncadeada< Musica > & getMusicas ()
  Obtém a referência para a lista encadeada de músicas da playlist.
- void adicionarMusica (Musica &musica)
  Adiciona uma música à playlist.
- void adicionarMusica (const Playlist &playlist)
  Adiciona todas as músicas de outra playlist à playlist atual.
- void removerMusica (const Musica &musica)
  Remove a música especificada da playlist.
- int removerMusica (const Playlist &playlist)
  Remove todas as músicas de outra playlist da playlist atual.
- Musica * buscarMusica (const Musica &musica)
  Procura uma música na playlist.
- void imprimirMusicas ()
  Imprime as músicas da playlist.
- bool operator== (Playlist &outra)
  Sobrecarga do operador de igualdade.
- Playlist operator+ (const Playlist &outra)
  Sobrecarga do operador de união de playlists.
- Playlist operator+ (Musica &musica)
  Sobrecarga do operador de união de playlist com música.
- Playlist operator- (const Playlist &outra)
  Sobrecarga do operador de diferença de playlists.
- Playlist operator- (Musica &musica)
  Sobrecarga do operador de diferença de playlist com música.
- Playlist & operator= (const Playlist &outra)
  Sobrecarga do operador de atribuição de playlist.
- Playlist & operator>> (Musica &musica)
  Sobrecarga do operador de extração de música de uma playlist.
- Playlist & operator<< (Musica &musica)
  Sobrecarga do operador de inserção de música em playlist.

Atributos Privados

- std::string **nome**

  Nome da playlist.
- ListaEncadeada< Musica > **musicas**

  Lista de músicas da playlist.

Amigos

- std::ostream & operator<< (std::ostream &os, const Playlist &playlist)

  Sobrecarga do operador de inserção de playlist.

4.4.1 Descrição detalhada

Classe que implementa uma playlist.

4.4.2 Construtores e Destrutores

4.4.2.1 Playlist() [1/3]

```
Playlist::Playlist ( )
```

Construtor padrão da classe Playlist.

Cria uma playlist com nome vazio.

4.4.2.2 Playlist() [2/3]

```
Playlist::Playlist (
            std::string nome )
```

Construtor da playlist que recebe seu nome.

Construtor da classe Playlist que recebe seu nome.

Parâmetros

nome	O nome da playlist.

Parâmetros

nome	O nome da playlist a ser criada.

4.4.2.3 Playlist() [3/3]

```
Playlist::Playlist (
```

```
            const Playlist & outra )
```

Construtor cópia da playlist.

Construtor de cópia da classe Playlist.

Parâmetros

outra	A outra playlist a ser copiada.

Cria uma nova playlist a partir de outra já existente, copiando seu nome e músicas.

Parâmetros

outra	A playlist a ser copiada.

4.4.2.4 ~Playlist()

```
Playlist::~Playlist ( )
```

Destrutor da playlist, que remove todas as músicas.

Destrutor da classe Playlist.

Libera a memória ocupada pelas músicas da playlist.

4.4.3 Documentação das funções

4.4.3.1 adicionarMusica() [1/2]

```
void Playlist::adicionarMusica (
            const Playlist & playlist )
```

Adiciona todas as músicas de outra playlist à playlist atual.

Parâmetros

playlist	A playlist de onde as músicas serão adicionadas.

Parâmetros

playlist	A playlist da qual as músicas serão adicionadas.

4.4.3.2 adicionarMusica() [2/2]

```
void Playlist::adicionarMusica (
            Musica & musica )
```

Adiciona uma música à playlist.

Parâmetros

musica	A música a ser adicionada.

Parâmetros

musica	A música a ser adicionada à playlist.

4.4.3.3 buscarMusica()

```
Musica * Playlist::buscarMusica (
            const Musica & musica )
```

Procura uma música na playlist.

Parâmetros

musica	A música a ser procurada.

Retorna

Um ponteiro para a música encontrada, ou nullptr se não encontrada.

Parâmetros

musica	A música a ser procurada na playlist.

Retorna

Um ponteiro para a música na playlist, ou nullptr se a música não estiver na playlist.

4.4.3.4 getMusicas()

```
ListaEncadeada< Musica > & Playlist::getMusicas ( )
```

Obtém a referência para a lista encadeada de músicas da playlist.

Retorna

A lista encadeada de músicas.

Retorna

A referência para a lista de músicas da playlist.

4.4.3.5 getNome()

```
const std::string & Playlist::getNome ( ) const
```

Obtém o nome da playlist.

Retorna

O nome da playlist.

Retorna

O nome da playlist como uma referência constante para uma string.

4.4.3.6 getTamanho()

```
size_t Playlist::getTamanho ( )
```

Obtém o tamanho da playlist (quantidade de músicas).

Obtém o tamanho da playlist.

Retorna

O tamanho da playlist.

Retorna

O número de músicas na playlist.

4.4.3.7 imprimirMusicas()

```
void Playlist::imprimirMusicas ( )
```

Imprime as músicas da playlist.

Imprime os títulos e autores de todas as músicas presentes na playlist.

4.4.3.8 operator+() [1/2]

```
Playlist Playlist::operator+ (
            const Playlist & outra )
```

Sobrecarga do operador de união de playlists.

Parâmetros

outra	A outra playlist a ser unida à playlist atual.

Retorna

Uma nova playlist contendo as músicas de ambas as playlists.

Cria uma nova playlist com o nome combinado das duas playlists de origem, contendo todas as músicas das playlists originais sem duplicações.

Parâmetros

outra	A playlist a ser unida com a playlist atual.

Retorna

Uma nova playlist resultante da união das playlists originais.

4.4.3.9 operator+() [2/2]

```
Playlist Playlist::operator+ (
            Musica & musica )
```

Sobrecarga do operador de união de playlist com música.

Parâmetros

musica	A música a ser adicionada à playlist atual.

Retorna

Uma nova playlist contendo as músicas da playlist atual mais a música adicionada.

Cria uma nova playlist a partir da playlist atual, adicionando uma música, desde que a música não esteja presente na playlist original.

Parâmetros

musica	A música a ser adicionada na nova playlist.

Retorna

Uma nova playlist resultante da adição da música à playlist original.

4.4.3.10 operator-() [1/2]

```
Playlist Playlist::operator- (
            const Playlist & outra )
```

Sobrecarga do operador de diferença de playlists.

Parâmetros

outra	A outra playlist a ser subtraída da playlist atual.

Retorna

Uma nova playlist contendo as músicas que estão na playlist atual e não na outra.

Cria uma nova playlist com o nome combinado das duas playlists de origem, contendo todas as músicas da playlist atual que não estão presentes na outra playlist.

Parâmetros

outra	A playlist a ser subtraída da playlist atual.

Retorna

Uma nova playlist resultante da diferença entre as playlists originais.

4.4.3.11 operator-() [2/2]

```
Playlist Playlist::operator- (
            Musica & musica )
```

Sobrecarga do operador de diferença de playlist com música.

Parâmetros

musica	A música a ser removida da playlist atual.

Retorna

Uma nova playlist contendo as músicas da playlist atual sem a música removida.

Cria uma nova playlist a partir da playlist atual, removendo a música especificada.

Parâmetros

musica	A música a ser removida da nova playlist.

Retorna

Uma nova playlist resultante da remoção da música da playlist original.

4.4.3.12 operator<<()

```
Playlist & Playlist::operator<< (
            Musica & musica )
```

Sobrecarga do operador de inserção de música em playlist.

Parâmetros

musica	A música a ser adicionada à playlist atual.

Retorna

A playlist atual após a adição da música.

Adiciona a música especificada à playlist atual, caso ela ainda não esteja presente.

Parâmetros

musica	A música a ser adicionada na playlist.

Retorna

Uma referência para a playlist atual após a adição da música.

4.4.3.13 operator=()

```
Playlist & Playlist::operator= (
            const Playlist & outra )
```

Sobrecarga do operador de atribuição de playlist.

Sobrecarga do operador de atribuição para a classe Playlist.

Parâmetros

outra	A outra playlist a ser copiada para a playlist atual.

Retorna

A playlist atual após a cópia.

Substitui o conteúdo da playlist atual pelo conteúdo da outra playlist fornecida.

Parâmetros

outra	A playlist a ser copiada.

Retorna

Uma referência para a playlist atual após a atribuição.

4.4.3.14 operator==()

```
bool Playlist::operator== (
            Playlist & outra )
```

Sobrecarga do operador de igualdade.

Sobrecarga do operador de igualdade para a classe Playlist.

Parâmetros

outra	A outra playlist a ser comparada.

Retorna

true se as playlists forem iguais, false caso contrário.

Verifica se duas playlists são iguais comparando seus nomes.

Parâmetros

outra	A playlist a ser comparada.

Retorna

true se as playlists têm o mesmo nome, false caso contrário.

4.4.3.15 operator>>()

```
Playlist & Playlist::operator>> (
            Musica & musica )
```

Sobrecarga do operador de extração de música de uma playlist.

Parâmetros

musica	O objeto que receberá a música extraída da playlist atual.

Retorna

A playlist atual após a extração da música.

Remove a última música da playlist atual e a armazena na variável 'musica'.

Parâmetros

musica	A música a ser extraída da playlist.

Retorna

Uma referência para a playlist atual após a extração.

4.4.3.16 removerMusica() [1/2]

```
void Playlist::removerMusica (
            const Musica & musica )
```

Remove a música especificada da playlist.

Parâmetros

musica	A música a ser removida.

Parâmetros

musica	A música a ser removida da playlist.

4.4.3.17 removerMusica() [2/2]

```
int Playlist::removerMusica (
            const Playlist & playlist )
```

Remove todas as músicas de outra playlist da playlist atual.

Remove da playlist todas as músicas presentes na outra playlist.

Parâmetros

playlist	A playlist de onde as músicas serão removidas.

Retorna

O número de músicas removidas.

Parâmetros

playlist	A playlist contendo as músicas a serem removidas.

Retorna

O número de músicas removidas da playlist atual.

4.4.4 Documentação dos símbolos amigos e relacionados

4.4.4.1 operator<<

```
std::ostream & operator<< (
```

```
std::ostream & os,
const Playlist & playlist )  [friend]
```

Sobrecarga do operador de inserção de playlist.

Parâmetros

os	O stream de saída.
playlist	A playlist a ser impressa no stream de saída.

Retorna

O stream de saída após a impressão da playlist.

Imprime as informações da playlist no stream de saída no formato ¨Nome - N músicas.¨.

Parâmetros

os	O stream de saída.
playlist	A playlist a ser exibida no stream.

Retorna

O stream de saída com as informações da playlist formatadas.

A documentação para essa classe foi gerada a partir dos seguintes arquivos:

- include/playlist.hpp
- src/playlist.cpp

Chapter 5

Arquivos

5.1 listaencadeada.hpp

```
00001 #ifndef LISTAENCADEADA_HPP
00002 #define LISTAENCADEADA_HPP
00003
00004 #include "musica.hpp"
00005 #include "no.hpp"
00006 #include <iostream>
00007
00012 template <typename T> class ListaEncadeada {
00013
00014 private:
00015   No<T> *cabeca;
00016   No<T> *cauda;
00017
00018 public:
00022   ListaEncadeada();
00023
00028   ListaEncadeada(const ListaEncadeada<T> &outra);
00029
00033   ~ListaEncadeada();
00034
00038   void limpar();
00039
00044   size_t getTamanho() const;
00045
00050   No<T> *getCabeca() const;
00051
00056   No<T> *getCauda() const;
00057
00062   void setCabeca(No<T> *cabeca);
00063
00068   void setCauda(No<T> *cauda);
00069
00074   void adicionar(T valor);
00075
00080   void adicionar(const ListaEncadeada<T> &outra);
00081
00087   T *buscarValor(T valor);
00088
00092   void removerFinal();
00093
00099   int removerValor(T valor);
00100
00105   void removerValor(ListaEncadeada<T> &outra);
00106
00110   void imprimir();
00111
00117   ListaEncadeada<T> operator+(ListaEncadeada<T> &outra);
00118
00124   const ListaEncadeada<T> &operator»(No<T> &no);
00125
00131   const ListaEncadeada<T> operator«(const No<T> &no);
00132 };
00133
00134 // construtor
00135 template <typename T> ListaEncadeada<T>::ListaEncadeada() {
00136   cabeca = nullptr;
00137   cauda = nullptr;
00138 }
```

```
00139
00140 template <typename T>
00141 ListaEncadeada<T>::ListaEncadeada(const ListaEncadeada<T> &outra) {
00142   cabeca = nullptr;
00143   cauda = nullptr;
00144
00145   adicionar(outra);
00146 }
00147
00148 // destrutor da lista encadeada, que remove todos os elementos.
00149 template <typename T> ListaEncadeada<T>::~ListaEncadeada() { limpar(); }
00150
00151 // remove todos os elementos da lista
00152 template <typename T> void ListaEncadeada<T>::limpar() {
00153   No<T> *anterior = nullptr;
00154   No<T> *atual = cabeca;
00155
00156   while (atual != nullptr) {
00157     anterior = atual;
00158     atual = atual->getProximo();
00159     delete anterior;
00160   }
00161   cabeca = nullptr;
00162   cauda = nullptr;
00163 }
00164
00165 // traz o tamanho da lista
00166 template <typename T> size_t ListaEncadeada<T>::getTamanho() const {
00167   size_t tamanho = 0;
00168   No<T> *atual = cabeca;
00169
00170   while (atual != nullptr) {
00171     atual = atual->getProximo();
00172     tamanho++;
00173   }
00174   return tamanho;
00175 }
00176
00177 // retorna a cabeça da lista
00178 template <typename T> No<T> *ListaEncadeada<T>::getCabeca() const {
00179   return cabeca;
00180 }
00181
00182 // retorna o final da lista
00183 template <typename T> No<T> *ListaEncadeada<T>::getCauda() const {
00184   return cauda;
00185 }
00186
00187 // altera o ponteiro incial da lista
00188 template <typename T> void ListaEncadeada<T>::setCabeca(No<T> *cabeca) {
00189   this->cabeca = cabeca;
00190 }
00191
00192 // altera o ponteiro final
00193 template <typename T> void ListaEncadeada<T>::setCauda(No<T> *cauda) {
00194   this->cauda = cauda;
00195 }
00196
00197 // coloca um elemento especifico ao final da lista
00198 template <typename T> void ListaEncadeada<T>::adicionar(T valor) {
00199   No<T> *novoNo = new No<T>(valor);
00200
00201   // Verifica se a lista é vazia
00202   if (cabeca == nullptr) {
00203     cabeca = novoNo;
00204     cauda = novoNo;
00205   } else {
00206     cauda->setProximo(novoNo);
00207     cauda = novoNo;
00208   }
00209 }
00210
00211 // adiciona uma lista ao final da outra lista.
00212 template <typename T>
00213 void ListaEncadeada<T>::adicionar(const ListaEncadeada<T> &outra) {
00214   No<T> *noAtual = outra.getCabeca();
00215
00216   while (noAtual != nullptr) {
00217     this->adicionar(noAtual->getValor());
00218     noAtual = noAtual->getProximo();
00219   }
00220 }
00221
00222 // procura um elemento da lista
00223 template <typename T> T *ListaEncadeada<T>::buscarValor(T valor) {
00224   No<T> *atual = cabeca;
00225
```

```
00226   while (atual != nullptr) {
00227     if (atual->getValor() == valor) {
00228       return &(atual->getValor());
00229     }
00230     atual = atual->getProximo();
00231   }
00232   return nullptr;
00233 }
00234
00235 // remove o elemento de um índice
00236 template <typename T> void ListaEncadeada<T>::removerFinal() {
00237   No<T> *atual = cabeca;
00238   No<T> *anterior = nullptr;
00239
00240   while (atual->getProximo() != nullptr) {
00241     anterior = atual;
00242     atual = atual->getProximo();
00243   }
00244
00245   if (anterior != nullptr) {
00246     anterior->setProximo(nullptr);
00247     cauda = anterior;
00248   } else {
00249     cabeca = nullptr;
00250     cauda = nullptr;
00251   }
00252
00253   delete atual;
00254 }
00255
00256 // remove o elemento especifico da lista.
00257 template <typename T> int ListaEncadeada<T>::removerValor(T valor) {
00258   No<T> *atual = cabeca;
00259   No<T> *anterior = nullptr;
00260
00261   while (atual != nullptr) {
00262     if (atual->getValor() == valor) {
00263       if (anterior != nullptr) {
00264         anterior->setProximo(atual->getProximo());
00265         if (atual == cauda) {
00266           cauda = anterior;
00267         }
00268       } else {
00269         cabeca = atual->getProximo();
00270         if (cabeca == nullptr) {
00271           cauda = nullptr;
00272         }
00273       }
00274       delete atual;
00275       return 1;
00276     }
00277     anterior = atual;
00278     atual = atual->getProximo();
00279   }
00280   return 0;
00281 }
00282
00283 // remove o elementos de uma lista da lista.
00284 template <typename T>
00285 void ListaEncadeada<T>::removerValor(ListaEncadeada<T> &outra) {
00286   No<T> *noAtual = outra.getCabeca();
00287
00288   while (noAtual != nullptr) {
00289     removerValor(noAtual->getValor());
00290     noAtual = noAtual->getProximo();
00291   }
00292 }
00293
00294 // imprime os elementos da lista utilizando recursão
00295 template <typename T> void ListaEncadeada<T>::imprimir() {
00296   imprimirAux(cabeca);
00297 }
00298
00299 // auxiliar para a função imprimir
00300 template <typename T> void imprimirAux(No<T> *atual) {
00301   if (atual != nullptr) {
00302     std::cout « atual->getValor() « std::endl;
00303     imprimirAux(atual->getProximo());
00304   }
00305 }
00306
00307 // sobrecarga de soma
00308 template <typename T>
00309 ListaEncadeada<T> ListaEncadeada<T>::operator+(ListaEncadeada<T> &outra) {
00310   ListaEncadeada<T> novaLista;
00311   novaLista.adicionar(*this);
00312   novaLista.adicionar(outra);
```

```
00313
00314   return novaLista;
00315 }
00316
00317 // sobrecarga de operador de extração
00318 template <typename T>
00319 const ListaEncadeada<T> &ListaEncadeada<T>::operator»(No<T> &no) {
00320   if (cabeca != nullptr) {
00321     no = *cauda;
00322     removerFinal();
00323   }
00324
00325   return *this;
00326 }
00327
00328 // sobrecarga de operador de inserção
00329 template <typename T>
00330 const ListaEncadeada<T> ListaEncadeada<T>::operator«(const No<T> &no) {
00331   adicionar(no.getValor());
00332   return *this;
00333 }
00334
00335 #endif
```

5.2 menu.hpp

```
00001 #include "listaencadeada.hpp"
00002 #include "musica.hpp"
00003 #include "no.hpp"
00004 #include "playlist.hpp"
00005 #include <iostream>
00006 #include <string>
00007
00014 int menuPrincipal(ListaEncadeada<Musica> &musicas,
00015                   ListaEncadeada<Playlist> &playlists);
00016
00022 void menuMusica(ListaEncadeada<Musica> &musicas,
00023                 ListaEncadeada<Playlist> &playlists);
00024
00029 void tocarMusicas(ListaEncadeada<Playlist> &playlists);
00030
00035 void menuPlaylist(ListaEncadeada<Playlist> &playlists);
00036
00042 void menuMusicaPlaylist(ListaEncadeada<Musica> &musicas,
00043                         ListaEncadeada<Playlist> &playlists);
```

5.3 musica.hpp

```
00001 #ifndef MUSICA_HPP
00002 #define MUSICA_HPP
00003
00004 #include <iostream>
00005 #include <string>
00006
00010 class Musica {
00011
00012 private:
00013   std::string titulo;
00014   std::string autor;
00015
00016 public:
00020   Musica();
00021
00028   Musica(std::string titulo, std::string autor = "");
00029
00035   std::string getTitulo();
00036
00042   std::string getAutor();
00043
00049   void setTitulo(std::string titulo);
00050
00056   void setAutor(std::string autor);
00057
00064   bool operator==(Musica &outra);
00065
00074   friend std::ostream &operator«(std::ostream &os, const Musica &musica);
00075 };
00076
00077 #endif
```

5.4 no.hpp

```
00001 #ifndef NO_HPP
00002 #define NO_HPP
00003
00008 template <typename T> class No {
00009
00010   T valor;
00011   No<T> *proximo;
00012
00013 public:
00017   No() = default;
00018
00023   No(const T &valor);
00024
00029   T &getValor();
00030
00035   No *getProximo();
00036
00041   void setValor(T valor);
00042
00047   void setProximo(No<T> *proximo);
00048 };
00049
00055 template <typename T> No<T>::No(const T &valor) {
00056   setValor(valor);
00057   setProximo(nullptr);
00058 }
00059
00065 template <typename T> T &No<T>::getValor() { return valor; }
00066
00072 template <typename T> No<T> *No<T>::getProximo() { return proximo; }
00073
00079 template <typename T> void No<T>::setValor(T valor) { this->valor = valor; }
00080
00086 template <typename T> void No<T>::setProximo(No<T> *proximo) {
00087   this->proximo = proximo;
00088 }
00089
00090 #endif
```

5.5 playlist.hpp

```
00001 #ifndef PLAYLIST_HPP
00002 #define PLAYLIST_HPP
00003
00004 #include "listaencadeada.hpp"
00005 #include "musica.hpp"
00006 #include "no.hpp"
00007 #include <string>
00008
00012 class Playlist {
00013
00014 private:
00015   std::string nome;
00016   ListaEncadeada<Musica> musicas;
00017
00018 public:
00022   Playlist();
00023
00029   Playlist(std::string nome);
00030
00036   Playlist(const Playlist &outra);
00037
00041   ~Playlist();
00042
00048   size_t getTamanho();
00049
00055   const std::string &getNome() const;
00056
00062   ListaEncadeada<Musica> &getMusicas();
00063
00069   void adicionarMusica(Musica &musica);
00070
00076   void adicionarMusica(const Playlist &playlist);
00077
00083   void removerMusica(const Musica &musica);
00084
00091   int removerMusica(const Playlist &playlist);
00092
00099   Musica *buscarMusica(const Musica &musica);
00100
00104   void imprimirMusicas();
```

```
00105
00112   bool operator==(Playlist &outra);
00113
00120   Playlist operator+(const Playlist &outra);
00121
00129   Playlist operator+(Musica &musica);
00130
00138   Playlist operator-(const Playlist &outra);
00139
00147   Playlist operator-(Musica &musica);
00148
00155   Playlist &operator=(const Playlist &outra);
00156
00163   Playlist &operator»(Musica &musica);
00164
00171   Playlist &operator«(Musica &musica);
00172
00180   friend std::ostream &operator«(std::ostream &os, const Playlist &playlist);
00181 };
00182
00183 #endif
```

5.6 arquivo.cpp

```
00001 #include <fstream>
00002 #include <iostream>
00003 #include <sstream>
00004 #include <string>
00005
00006 #include "../include/listaencadeada.hpp"
00007 #include "../include/menu.hpp"
00008 #include "../include/musica.hpp"
00009 #include "../include/no.hpp"
00010 #include "../include/playlist.hpp"
00011
00020 void parseFile(std::ifstream &in_file, ListaEncadeada<Musica> &songs,
00021                ListaEncadeada<Playlist> &playlists) {
00022   std::string linha;
00023   while (std::getline(in_file, linha)) {
00024     std::istringstream iss{linha};
00025     std::string playlist_titulo;
00026     std::getline(iss, playlist_titulo, ';');
00027
00028     Playlist playlist{playlist_titulo};
00029     playlists.adicionar(playlist);
00030
00031     std::string song_string;
00032     while (std::getline(iss, song_string, ',')) {
00033       std::istringstream iss_song{song_string};
00034       std::string musica_titulo;
00035       std::string musica_autor;
00036
00037       std::getline(iss_song, musica_titulo, ':');
00038       std::getline(iss_song, musica_autor);
00039
00040       Musica Musica{musica_titulo, musica_autor};
00041       songs.adicionar(Musica);
00042
00043       playlists.buscarValor(playlist)->adicionarMusica(Musica);
00044     }
00045   }
00046 }
00047
00055 void writeFile(std::ofstream &out_file, ListaEncadeada<Musica> &musicas,
00056                ListaEncadeada<Playlist> &playlists) {
00057   auto plRunner = playlists.getCabeca();
00058
00059   while (plRunner != nullptr) {
00060     auto currPl = plRunner->getValor();
00061
00062     out_file « currPl.getNome() « ';';
00063
00064     auto songRunner = currPl.getMusicas().getCabeca();
00065
00066     while (songRunner != nullptr) {
00067       auto currSong = songRunner->getValor();
00068
00069       out_file « currSong.getTitulo() « ':';
00070       out_file « currSong.getAutor();
00071
00072       if (songRunner->getProximo() != nullptr) {
00073         out_file « ',';
00074       }
```

```
00075
00076       songRunner = songRunner->getProximo();
00077     }
00078
00079     out_file « '\n';
00080     plRunner = plRunner->getProximo();
00081   }
00082 }
```