

Gerenciador de Músicas

v1.0

Gerado por Doxygen 1.9.6

Chapter 1

README

Grupo: Erick Marques Oliveira Azevedo - Matricula 20210047901

1.0.1 Como rodar o programa:

1.0.1.1 Usando o WSL/Bash:

```
# Na pasta do programa compile o programa usando o CMake:
$ cmake .
$ make

# Em seguida execute o arquivo gerado com o seguinte comando:
$ ./music
```

Os comandos disponíveis neste programa são:

1.0.2 Comando de ajuda e encerramento do programa:

```
# help - Comando que exibe a lista de comandos.
# quit - Comando que encerra o programa.
```

1.0.3 Comandos para gerenciamento de músicas:

```
# add - Comando que adiciona uma música.
# del - Comando que deleta uma música previamente armazenada.
# list - Comando que lista todas as músicas armazenadas.
# search - Comando que busca uma música específica na lista de músicas armazenadas.
```

1.0.4 Comandos para gerenciamento de playlists:

```
# addp - Comando que adiciona uma playlist.
# delp - Comando que elimina uma playlist previamente armazenada.
# listp - Comando que lista todas as playlists armazenadas.
```

1.0.5 Comandos para reprodução de música nas playlists:

```
# playp - Comando para reproduzir uma playlist.  
# playn - Comando para que a próxima música da playlist seja reproduzida.  
# playb - Comando para que a música anterior da playlist seja reproduzida.  
# plays - Comando para que a reprodução de músicas pare.
```

1.0.6 Comandos para gerenciamento de música nas playlists:

```
# addmp - Comando que adiciona uma música em uma playlist.  
# delmp - Comando que remove uma música previamente armazenada em uma playlist.  
# mmp = Comando que move músicas em uma playlist.  
# listmp - Comando que lista músicas em uma playlist.  
# copymp - Comando que cria uma cópia de uma playlist existente.  
# mergep - Comando que cria uma nova playlist união de duas playlists.  
# mergep- - Comando que cria uma nova playlist, removendo as músicas da segunda playlist da primeira.  
# savetf - Salvar uma playlist em um arquivo "Playlists.txt".
```

1.0.7 Comandos para testes das funções:

```
# addtp - Adiciona algumas músicas e playlists pré-definidas.  
# otest - Testa os métodos sobrecarregados.
```

1.0.8 Exemplo de utilização do programa:

1.0.8.1 Iniciando o programa

```
$ ./music
```

1.0.8.2 Tela inicial do programa:

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
Digite um comando:
```

1.0.8.3 Comando "help" para exibir todos os comandos:

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
Digite um comando: help
```

Comandos para gerenciamento de músicas:

```
add - Adicionar uma música  
del - Remover uma música  
list - Listar todas as músicas  
search - Buscar uma música
```

Comandos para gerenciamento de playlists:

```
addp - Adicionar uma playlist  
delp - Remover uma playlist  
listp - Listar todas as playlists
```

Comandos para reproduzir músicas de playlists:

```
playp - Comece a tocar uma playlist
```

```
playn - Toque a próxima música de uma playlist.
playb - Volte uma música.
plays - Pare a reprodução de músicas.
```

Comandos para gerenciamento de músicas em playlists:

```
addmp - Adicionar música a uma playlist
delmp - Remover música de uma playlist
mmp - Mover música numa playlist
listmp - Listar músicas de uma playlist
copyp - Criar uma cópia de uma playlist existente.
mergep - Unir duas playlists em uma única playlist
mergep- - Remover músicas da segunda playlist da primeira e retornar uma nova playlist
```

```
savetf - Salvar uma playlist em um arquivo
```

Comandos para teste:

```
addtp - Adicionar músicas e playlists pré-definidas ao programa.
otest - Teste de métodos sobrecarregados.
```

```
quit - Encerrar o programa.
```

1.0.8.4 Adicionando uma música

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: add
```

```
-----
Título da música: Shiny and New
Nome do artista: Lonely Bunker
```

```
Música adicionada com sucesso!
```

1.0.8.5 Removendo uma música previamente adicionada

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: del
```

```
-----
1 - Shiny and New - Lonely Bunker
```

```
Insira o índice da música a ser removida (consulte a lista de músicas): 1
A música Shiny and New - Lonely Bunker foi deletada.
```

1.0.8.6 Listando todas as músicas adicionadas

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: list
```

```
-----
Músicas armazenadas:
```

```
1 - Shiny and New - Lonely Bunker
2 - A Sky Full Of Stars - Coldplay
3 - Complicated - Avril Lavigne
4 - Careless Whisper - George Michael
```

1.0.8.7 Verificando se uma música já foi adicionada

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: search
```

```
-----
Título da música: A Sky Full Of Stars
Nome do artista: Coldplay
```

```
A música A Sky Full Of Stars - Coldplay está armazenada!
```

1.0.8.8 Adicionando uma playlist

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
  
Digite um comando: addp
```

```
-----  
  
Nome da playlist: Favoritas  
Playlist criada com sucesso!
```

1.0.8.9 Eliminando uma playlist

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
  
Digite um comando: delp  
  
-----  
  
Playlists registradas:  
1 - Favoritas  
  
Insira o índice da playlist a ser removida: 1  
Remoção concluída com sucesso.
```

1.0.8.10 Listar todas as playlists

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
  
Digite um comando: listp  
  
-----  
  
Playlists armazenadas atualmente:  
  
1 - Pop  
2 - Eletrônica  
3 - Rock  
4 - Lo-Fi
```

1.0.8.11 Iniciando a reprodução de uma playlist

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
  
Digite um comando: playp  
  
-----  
  
Playlists disponíveis:  
1 - Pop  
2 - Eletrônica  
3 - Rock  
4 - Lo-Fi  
Insira o índice da playlist desejada: 3  
  
1 - Take It - Avril Lavigne  
2 - Paradise City - Guns N Roses  
3 - Could Have Been Me - The Struts  
4 - Hot Night Crash - Sahara Hotnights  
5 - Complicated - Avril Lavigne  
  
Insira o índice da música que será tocada (consulte a lista de músicas): 1  
Tocando agora: Take It - Avril Lavigne
```

1.0.8.12 Tocar a próxima música na playlist

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
  
Digite um comando: playn  
  
-----  
  
Tocando agora: Paradise City - Guns N Roses
```

1.0.8.13 Tocar a música anterior na playlist

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: playb

-----

Tocando agora: Take It - Avril Lavigne
```

1.0.8.14 Parar a reprodução de músicas

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: plays

-----

Parando a reprodução!
```

1.0.8.15 Adicionar música a uma playlist

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: addmp

-----

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
Insira o índice da playlist desejada (enumerada acima): 3
Playlist selecionada: Rock

Título da música: Complicated
Nome do artista: Avril Lavigne
Em qual posição deseja adicionar (consulte a lista de músicas): 1

Música adicionada à playlist 'Rock'
```

1.0.8.16 Eliminando uma música de uma playlist

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: delmp

-----

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
Insira o índice da playlist desejada: 1
Playlist selecionada: Pop

1 - Shiny and New - Lonely Bunker
2 - Careless Whisper - George Michael
3 - Call Me Maybe - Carly Rae Jepsen
Insira o índice da música a ser removida (consulte a lista de músicas): 1

Música removida da playlist 'Pop'
```

1.0.8.17 Movendo músicas em uma playlist

```
----- Tela Inicial -----
```

```
Para uma lista de comandos digite 'help'

Digite um comando: mmp

-----

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
Insira o índice da playlist desejada: 3
Playlist selecionada: Rock

Músicas nesta playlist:

1 - Take It - Avril Lavigne
2 - Paradise City - Guns N Roses
3 - Could Have been Me - The Struts
4 - Hot Night Crash - Sahara Hotnights
5 - Complicated - Avril Lavigne

Insira o índice da música a ser movida: 3
Insira o índice da posição para qual deseja movê-la: 1

Posição alterada com sucesso.
```

1.0.8.18 Listando todas as músicas em uma playlist

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: listmp

-----

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
Insira o índice da playlist desejada: 3

Músicas da playlist 'Rock':

1 - Could Have Been Me - The Struts
2 - Take It - Avril Lavigne
3 - Paradise City - Guns N Roses
4 - Hot Night Crash - Sahara Hotnights
5 - Complicated - Avril Lavigne
```

1.0.8.19 Criando uma playlist cópia usando uma playlist já existente

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: copyp

-----

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
Insira o índice da playlist desejada: 3

Digite o nome da nova playlist: Cópia de Rock

Playlist criada com sucesso!
```

1.0.8.20 Unindo duas playlists em uma nova playlist

```
----- Tela Inicial -----
```

```
Para uma lista de comandos digite 'help'

Digite um comando: mergep

-----

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
5 - Cópia de Rock
Insira o índice da primeira playlist desejada: 3

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
5 - Cópia de Rock
Insira o índice da segunda playlist desejada: 1

Digite o nome da nova playlist: Pop-Rock

Playlist criada com sucesso!
```

1.0.8.21 Apagando as músicas de uma playlist em outra playlist

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: mergep-

-----

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
5 - Cópia de Rock
6 - Pop-Rock
Insira o índice da primeira playlist desejada: 6

Playlists disponíveis:
1 - Pop
2 - Eletrônica
3 - Rock
4 - Lo-Fi
5 - Cópia de Rock
6 - Pop-Rock
Insira o índice da segunda playlist desejada: 1

Digite o nome da nova playlist: Rock 2

Playlist criada com sucesso!
```

1.0.8.22 Salvando uma playlist em um arquivo

```
----- Tela Inicial -----
Para uma lista de comandos digite 'help'

Digite um comando: savetf

Escolha a playlist que você quer armazenar:
1 - Rock
2 - Pop
3 - Eletrônica
4 - Lo-Fi
5 - Cópia de Rock
6 - Pop-Rock
7 - Rock 2
Insira o índice da playlist desejada: 7

-> Output em Playlists.txt:

Rock 2;Could Have Been Me:The Struts,Take It:Avril Lavigne,Paradise City:Guns N Roses,Hot Night Crash:Sahara
Hotnights,Complicated:Avril Lavigne
```

1.0.8.23 Adicionando músicas e playlists pré-definidas

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
Digite um comando: addtp
```

```
Playlist criada com sucesso!  
Playlist criada com sucesso!  
Músicas adicionadas com sucesso!
```

1.0.8.24 Encerrando o programa

```
----- Tela Inicial -----  
Para uma lista de comandos digite 'help'  
Digite um comando: quit  
-----  
Encerrando o programa.
```

A documentação pode ser encontrada em pdf no arquivo "Documentação.pdf" na pasta raiz do programa ou em html na pasta "/docs/html/index.html".

Chapter 2

Índice dos Componentes

2.1 Lista de Classes

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

LinkedList	??
ListOfPlaylists	??
no	??
no_	??
Playlist	??
Song	??

Chapter 3

Índice dos Arquivos

3.1 Lista de Arquivos

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

include/listaLigada.h	??
include/listaPlaylists.h	??
include/musica.h	??
include/playlist.h	??
include/utilitarios.h	??
src/listaLigada.cpp	
Funções necessárias para as listas ligadas de músicas	??
src/listaPlaylists.cpp	
Funções necessárias para as listas ligadas de playlists	??
src/main.cpp	
Projeto que organiza músicas e playlists em listas ligadas	??
src/musica.cpp	
Funções que definem e exibem informações das músicas	??
src/playlist.cpp	
Funções que definem, exibem e permitem o funcionamento das playlists	??
src/utilitarios.cpp	
Função que transforma caracteres maiúsculos em minúsculos	??

Chapter 4

Classes

4.1 Referência da Classe LinkedList

```
#include <listaLigada.h>
```

Diagrama de colaboração para LinkedList:

4.2 Referência da Classe ListOfPlaylists

```
#include <listaPlaylists.h>
```

Diagrama de colaboração para ListOfPlaylists:

Membros Públicos

- `ListOfPlaylists ()`
- `~ListOfPlaylists ()`
- `size_t getSize ()`
- `Playlist * getPlaylist (size_t pos)`
- `Playlist * searchPlaylist (std::string searchName)`
- `void insertPlaylist (Playlist *value)`
- `void removePlaylist (size_t pos)`
- `void removeFromAll (Song target)`
- `void display ()`

Função que exibe todas as playlists na lista ligada.

Atributos Privados

- `no_ * head`
- `no_ * tail`
- `size_t size`

4.2.1 Descrição detalhada

Definição na linha 14 do arquivo [listaPlaylists.h](#).

4.2.2 Construtores e Destrutores

4.2.2.1 ListOfPlaylists()

```
ListOfPlaylists::ListOfPlaylists ( )
```

< Inicializando head e tail com valores nulos e tamanho 0. Caso os nós da lista ainda existam na memória, eles são desalocados pela função abaixo.

Definição na linha 15 do arquivo [listaPlaylists.cpp](#).

4.2.2.2 ~ListOfPlaylists()

```
ListOfPlaylists::~~ListOfPlaylists ( )
```

Adquire o tamanho da lista de playlists.

Definição na linha 21 do arquivo [listaPlaylists.cpp](#).

4.2.3 Documentação das funções

4.2.3.1 display()

```
void ListOfPlaylists::display ( )
```

Função que exibe todas as playlists na lista ligada.

Definição na linha 148 do arquivo [listaPlaylists.cpp](#).

```
00148 {  
00149     no_* temp = head;  
00150     size_t i = 1;  
00151     string nome = "";  
00152     while (temp != nullptr) {  
00153         nome = temp->data->getName();  
00154         cout << i << " - " << nome << endl;  
00155         temp = temp->next;  
00156         ++i;  
00157     }  
00158 }
```

4.2.3.2 getPlaylist()

```
Playlist * ListOfPlaylists::getPlaylist (   
    size_t pos )
```

Essa função percorre a lista até a posição passada por parâmetro e obtém o ponteiro da playlist correspondente.

Parâmetros

<i>pos</i>	é o índice da posição escolhida (inicia-se em 1).
------------	---

Retorna

o ponteiro para a playlist, caso a posição desejada esteja dentro do tamanho da lista, ou nullptr caso não esteja.

O if abaixo retorna nullptr se a posição não for válida.

Caso seja válida entretanto, a função retorna o ponteiro para a playlist correspondente.

Definição na linha 43 do arquivo `listaPlaylists.cpp`.

```
00043                                     {
00045     if (pos < 1 || pos > size) {
00046         return nullptr;
00047     } else {
00049         no_* temp = head;
00050         for (size_t i = 1; i < pos; ++i) {
00051             temp = temp->next;
00052         }
00053         return temp->data;
00054     }
00055 }
```

4.2.3.3 getSize()

```
size_t ListOfPlaylists::getSize ( )
```

Definição na linha 35 do arquivo `listaPlaylists.cpp`.

```
00035                                     {
00036     return size;
00037 }
```

4.2.3.4 insertPlaylist()

```
void ListOfPlaylists::insertPlaylist (
    Playlist * value )
```

Essa função cria e insere no fim da lista um nó que armazena a playlist passada por argumento

Parâmetros

<i>value</i>	é um ponteiro para um objeto do tipo <code>Playlist</code> .
--------------	--

Definição na linha 77 do arquivo `listaPlaylists.cpp`.

```
00077                                     {
00078     if (searchPlaylist(value->getName()) != nullptr) {
00079         cout << "Uma playlist com esse nome já existe!" << endl << endl;
00080     } else {
00081         no_* temp = new no_;
00082         temp->data = value;
```

```

00083     temp->next = nullptr;
00084     if (head == nullptr) {
00085         head = temp;
00086         tail = temp;
00087         temp = nullptr;
00088     } else {
00089         tail->next = temp;
00090         tail = temp;
00091     }
00092     ++size;
00093     cout << endl << "Playlist criada com sucesso!" << endl;
00094 }
00095 }

```

4.2.3.5 removeFromAll()

```

void ListOfPlaylists::removeFromAll (
    Song target )

```

Percorre todas as playlists do sistema e elimina de todas a música passada por parâmetro.

Parâmetros

<i>target</i>	é um objeto do tipo Song com a música a ser removida.
---------------	---

Definição na linha 132 do arquivo [listaPlaylists.cpp](#).

```

00132     {
00133     no_* temp = head;
00134     //Loop que percorre todas as playlists armazenadas.
00135     while (temp != nullptr) {
00136         LinkedList* songs = temp->data->getSongs(); //Salvando a lista de músicas atual.
00137         //If que Verifica se a musica está na lista
00138         if (songs->search(target) != nullptr) {
00139             size_t pos = songs->getPosition(target); //Obtém o ID da música.
00140             songs->removePosition(pos); //elimina a música da lista.
00141         }
00142         temp = temp->next;
00143     }
00144 }

```

4.2.3.6 removePlaylist()

```

void ListOfPlaylists::removePlaylist (
    size_t pos )

```

Essa função deleta o nó da posição escolhida e recria o link entre seu antecessor e sucessor.

Parâmetros

<i>pos</i>	é o índice da posição escolhida, nesse caso começa por 1.
------------	---

Caso a posição escolhida seja a primeira ou maior que a última esse if e else fazem o devido tratamento.

Definição na linha 101 do arquivo [listaPlaylists.cpp](#).

```

00101     {
00102     no_* pre = nullptr;
00103     no_* cur = nullptr;

```

```

00104
00105     cur = head;
00107     if (pos == 1) {
00108         head = head->next;
00109     } else if (pos < size) {
00110         for (size_t i = 1; i < pos; ++i) {
00111             pre = cur;
00112             cur = cur->next;
00113         }
00114         pre->next = cur->next;
00115     } else {
00116         while (cur->next != nullptr) {
00117             pre = cur;
00118             cur = cur->next;
00119         }
00120         tail = pre;
00121         pre->next = nullptr;
00122     }
00123     --size; //Reduzindo o tamanho da lista após a operação.
00124     delete cur->data;
00125     delete cur;
00126 }

```

4.2.3.7 searchPlaylist()

```

Playlist * ListOfPlaylists::searchPlaylist (
    std::string searchName )

```

Essa função percorre a lista procurando a playlist com o nome passado pelo parâmetro e obtém o ponteiro correspondente.

Parâmetros

<i>searchName</i>	é a playlist a ser buscada.
-------------------	-----------------------------

Retorna

o ponteiro para a playlist, caso a busca tenha sucesso, ou nullptr caso contrário.

Transforma os caracteres do nome da playlist em minúsculos (sem alterar os valores originais) e os compara, em seguida, retorna o ponteiro para o nó correspondente ou nullptr caso não encontre.

Definição na linha 61 do arquivo `listaPlaylists.cpp`.

```

00061
00062     no_* temp = head;
00063     while (temp != nullptr) {
00064         if ( toLowercase(temp->data->getName()) == toLowercase(searchName) ) {
00065             return temp->data;
00066         }
00067         temp = temp->next;
00068     }
00069     return nullptr;
00070 }
00071
00072 }

```

4.2.4 Atributos

4.2.4.1 head

```
no_* ListOfPlaylists::head [private]
```

Definição na linha 16 do arquivo [listaPlaylists.h](#).

4.2.4.2 size

```
size_t ListOfPlaylists::size [private]
```

Definição na linha 18 do arquivo [listaPlaylists.h](#).

4.2.4.3 tail

```
no_* ListOfPlaylists::tail [private]
```

Definição na linha 17 do arquivo [listaPlaylists.h](#).

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

- [include/listaPlaylists.h](#)
- [src/listaPlaylists.cpp](#)

4.3 Referência da Estrutura no

```
#include <listaLigada.h>
```

Diagrama de colaboração para no:

Atributos Públicos

- [Song data](#)
- [no * next](#)

4.3.1 Descrição detalhada

Definição na linha 7 do arquivo [listaLigada.h](#).

4.3.2 Atributos

4.3.2.1 data

```
Song no::data
```

Definição na linha 8 do arquivo [listaLigada.h](#).

4.3.2.2 next

```
no* no::next
```

Definição na linha 9 do arquivo [listaLigada.h](#).

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

- [include/listaLigada.h](#)

4.4 Referência da Estrutura no_

```
#include <listaPlaylists.h>
```

Diagrama de colaboração para no_:

Atributos Públicos

- [Playlist * data](#)
- [no_ * next](#)

4.4.1 Descrição detalhada

Definição na linha 8 do arquivo [listaPlaylists.h](#).

4.4.2 Atributos

4.4.2.1 data

```
Playlist* no_::data
```

Definição na linha 9 do arquivo [listaPlaylists.h](#).

4.4.2.2 next

```
no_* no_::next
```

Definição na linha 10 do arquivo [listaPlaylists.h](#).

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

- [include/listaPlaylists.h](#)

4.5 Referência da Classe Playlist

```
#include <playlist.h>
```

Diagrama de colaboração para Playlist:

Membros Públicos

- [Playlist](#) ()
- [~Playlist](#) ()
- [Playlist](#) (const [Playlist](#) &old)
- [Playlist operator+](#) ([Playlist](#) &secondPlaylist)
- [Playlist operator+](#) ([Song](#) &toAdd)
- [Playlist operator-](#) ([Playlist](#) &secondPlaylist)
- [Playlist operator-](#) ([Song](#) &toRemove)
- void [operator>>](#) ([Song](#) *&lastSong)
- void [operator<<](#) ([Song](#) *&newSong)
- [LinkedList](#) * [getSongs](#) ()
- string [getName](#) ()
- void [setName](#) (string _name)
- void [insertSong](#) (size_t pos, [Song](#) value)
- void [removeSong](#) (size_t pos)
- void [moveSong](#) (size_t start, size_t end)
- void [insertSong](#) ([Playlist](#) &toInsert)
- size_t [removeSong](#) ([Playlist](#) &toRemove)
- [no](#) * [playNext](#) ()
- void [displayAllSongs](#) ([no](#) *current)
- void [saveAlltofile](#) ([no](#) *current, ofstream &myfile)
Função recursiva que adiciona uma playlist a um arquivo "Playlist.txt".
- void [displayOne](#) ([no](#) *current, int pos)

Atributos Privados

- string [name](#)
- [LinkedList](#) * [songs](#)
- [no](#) * [playing](#)
- size_t [count](#)

4.5.1 Descrição detalhada

Definição na linha 11 do arquivo [playlist.h](#).

4.5.2 Construtores e Destrutores

4.5.2.1 Playlist() [1/2]

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

< Aloca a lista de músicas e inicializa os atributos necessários. Libera a memória da lista de músicas

Definição na linha 14 do arquivo [playlist.cpp](#).

4.5.2.2 ~Playlist()

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

Retorna as músicas de uma playlist

Definição na linha 20 do arquivo [playlist.cpp](#).

4.5.2.3 Playlist() [2/2]

```
Playlist::Playlist (
    const Playlist & old )
```

Essa função copia as músicas da playlist atual, insere a música passada por parâmetro e retorna uma nova playlist

Definição na linha 43 do arquivo [playlist.cpp](#).

4.5.3 Documentação das funções

4.5.3.1 displayAllSongs()

```
void Playlist::displayAllSongs (
    no * current )
```

Função recursiva que exibe todas as músicas que formam a playlist

Parâmetros

<i>current</i>	ponteiro para o nó atual contendo a música a que será exibida.
----------------	--

Condição de parada

Exibe a música atual

Chamada recursiva para que todas as músicas sejam exibidas

Definição na linha 202 do arquivo [playlist.cpp](#).

```
00202                                     {
00204     if (current == nullptr) {
00205         count = 1;
00206         return;
00207     }
00208     cout << count << " - " << current->data.getTitle() << " - " << current->data.getArtist() << endl;
00209     ++count;
00210     displayAllSongs(current->next);
00211 }
```

4.5.3.2 displayOne()

```
void Playlist::displayOne (
    no * current,
    int pos )
```

Função recursiva que exibe a música que o usuário deseja, usando seu índice.

Parâmetros

<i>current</i>	ponteiro para o nó atual contendo a música a que será exibida.
<i>pos</i>	valor do índice.

Exibe a música atual

Definição na linha 257 do arquivo [playlist.cpp](#).

```
00257                                     {
00258     if (current == nullptr) {
00259         count = 1;
00260         return;
00261     }
00262     if(count == pos){
00263     cout << current->data.getTitle() << " - " << current->data.getArtist() << endl;
00264     }
00265     ++count;
00266     displayOne(current->next, pos);
00267 }
```

4.5.3.3 getName()

```
string Playlist::getName ( )
```

Define o nome de uma playlist

Definição na linha 28 do arquivo [playlist.cpp](#).

4.5.3.4 getSongs()

```
LinkedList * Playlist::getSongs ( )
```

Retorna o nome de uma playlist

Definição na linha 24 do arquivo [playlist.cpp](#).

4.5.3.5 insertSong() [1/2]

```
void Playlist::insertSong (
    Playlist & toInsert )
```

Versão sobrecarregada do método de inserção, que insere na playlist atual todas as músicas da playlist passada por argumento.

Parâmetros

<i>toInsert</i>	é referência de um objeto do tipo playlist.
-----------------	---

Verificação para ver se a nova playlist não está vazia

Inserindo as músicas na playlist atual por meio de sobrecarga

Definição na linha 164 do arquivo [playlist.cpp](#).

```
00164 {
00166     if (toInsert.getSongs()->getSize() < 1) {
00167         return;
00168     } else {
00169         songs->insertEnd(*toInsert.getSongs());
00170     }
00171 }
```

4.5.3.6 insertSong() [2/2]

```
void Playlist::insertSong (
    size_t pos,
    Song value )
```

Chama o método da lista para inserir a música na playlist de acordo com a posição.

Parâmetros

<i>pos</i>	é o índice da posição escolhida (começa em 1).
<i>value</i>	é um objeto do tipo Song .

Cada vez que uma nova música for inserida, playing recebe o head da lista

Definição na linha 125 do arquivo [playlist.cpp](#).

```

00125                                     {
00126     songs->insertPosition(pos, value);
00127     playing = songs->getHead();
00128 }

```

4.5.3.7 moveSong()

```

void Playlist::moveSong (
    size_t start,
    size_t end )

```

Move uma música de acordo com as posições passadas por parâmetro

Parâmetros

<i>start</i>	Índice da posição inicial da música.
<i>end</i>	Índice da nova posição.

Acessando o ID da música que será movida

Removendo a música da posição atual

Inserindo a música na nova posição

Cada vez que uma música for movida, playing recebe o head da lista

Definição na linha 143 do arquivo [playlist.cpp](#).

```

00143                                     {
00144     if (start != end) {
00145         no* target = songs->getno(start);
00146         Song value = target->data;
00147         songs->removePosition(start);
00148         if (end < start) {
00149             songs->insertPosition(end, value);
00150         } else {
00151             songs->insertPosition(end + 1, value);
00152         }
00153         playing = songs->getHead();
00154     }
00155 }
00156 }
00157 }
00158 }

```

4.5.3.8 operator+() [1/2]

```

Playlist Playlist::operator+ (
    Playlist & segPlaylist )

```

Essa função copia todas as músicas da playlist passada por referência para a playlist que está sendo usada atualmente Criando uma nova playlist vazia

Inserir as músicas da playlist atual na playlist resultante

Inserir as músicas da segunda playlist na playlist resultante

Retorna a playlist resultante

Definição na linha 36 do arquivo [playlist.cpp](#).

4.5.3.9 operator+() [2/2]

```
Playlist Playlist::operator+ (
    Song & toAdd )
```

Criando uma nova playlist vazia

Inserindo músicas da playlist atual na playlist resultante

Inserindo a música do parâmetro no final da playlist

Retornando a playlist final

Definição na linha 55 do arquivo [playlist.cpp](#).

```
00055 {
00056     Playlist resultante;
00057     resultante.insertSong(*this);
00058     resultante.songs->insertEnd(toAdd);
00059     return resultante;
00060 }
```

4.5.3.10 operator-() [1/2]

```
Playlist Playlist::operator- (
    Playlist & segPlaylist )
```

Cria uma playlist nova contendo todas as músicas da playlist atual que não estão na playlist passada por referência.

Parâmetros

<i>segPlaylist</i>	referência de objeto do tipo Playlist .
--------------------	---

Retorna

a playlist resultante.

Cria uma playlist resultante a partir da playlist atual

Loop while que percorre a segunda playlist

Procurando a posição da música na playlist resultante

Retornando a playlist resultante

Definição na linha 67 do arquivo [playlist.cpp](#).

```
00067 {
00068     Playlist resultante(*this);
00069     no* temp = segPlaylist.getSongs()->getHead();
00070     while (temp != nullptr) {
00071         size_t pos = resultante.getSongs()->getPosition(temp->data);
00072         if (pos > 0) {
00073             resultante.removeSong(pos);
00074         }
00075         temp = temp->next;
00076     }
00077     return resultante;
00078 }
00079
00080 }
```

4.5.3.11 operator-() [2/2]

```
Playlist Playlist::operator- (
    Song & toRemove )
```

Copia as músicas da playlist atual, remove a música passada por parâmetro e retorna uma playlist final.

Parâmetros

<i>toRemove</i>	é a referência de objeto do tipo Song .
-----------------	---

Retorna

é a playlist resultante.

Criando a playlist resultante a partir da playlist atual

Obtendo a posição da música na playlist final

Se a musica existir, na playlist ela é removida

Retornando a playlist reslutante

Definição na linha 86 do arquivo [playlist.cpp](#).

```
00086 {
00087     Playlist resultante(*this);
00088     size_t pos = resultante.getSongs()->getPosition(toRemove);
00090     if (pos > 0) {
00091         resultante.removeSong(pos);
00092     }
00093     return resultante;
00094 }
```

4.5.3.12 operator<<()

```
void Playlist::operator<< (
    Song *& newSong )
```

Adiciona uma música passada por argumento no fim da playlist.

Parâmetros

<i>newSong</i>	é a música a ser inserida.
----------------	----------------------------

Definição na linha 113 do arquivo [playlist.cpp](#).

```
00113 {
00114     if (newSong == nullptr) {
00115         return;
00116     } else {
00117         songs->insertEnd(*newSong);
00118     }
00119 }
```

4.5.3.13 operator>>()

```
void Playlist::operator>> (
    Song *& lastSong )
```

Extrai a última música da playlist e atribui seus valores a música recebida como argumento

Parâmetros

<i>lastSong</i>	recebe os valores da música extraída.
-----------------	---------------------------------------

Extraindo o ultimo nó da lista de músicas

Guardando a música desse nó

Definição na linha 99 do arquivo `playlist.cpp`.

```
00099 {
00100     if (songs->getSize() > 0) {
00101         no* temp = new no;
00102         *songs » temp;
00103         *lastSong = temp->data;
00104         delete temp;
00105     } else {
00106         lastSong = nullptr;
00107     }
00108 }
```

4.5.3.14 playNext()

```
no * Playlist::playNext ( )
```

4.5.3.15 removeSong() [1/2]

```
size_t Playlist::removeSong (
    Playlist & toRemove )
```

Versão sobrecarregada do método de remoção, que remove da playlist atual todas as músicas da playlist passada por parâmetro.

Parâmetros

<i>toRemove</i>	é referência de um objeto do tipo playlist.
-----------------	---

Retorna

quantos elementos foram removidos.

Verificação para ver se a nova playlist não está vazia

Obtendo a posição da música na playlist atual

Removendo a música da playlist atual

Retornando a quantidade de elementos removidos

Definição na linha 178 do arquivo `playlist.cpp`.

```
00178 {
00180     if (toRemove.getSongs()->getSize() < 1) {
00181         return 0;
00182     } else {
00183         size_t removed = 0;
00184         no* temp = toRemove.getSongs()->getHead();
00185         while (temp != nullptr) {
00186             size_t pos = getSongs()->getPosition(temp->data);
00187             if (pos > 0) {
00188                 removeSong(pos);
00189                 ++removed;
00190             }
00191             temp = temp->next;
00192         }
00193         return removed;
00194     }
00195 }
00196 }
```

4.5.3.16 removeSong() [2/2]

```
void Playlist::removeSong (
    size_t pos )
```

Chama o método da lista para remover uma música da playlist baseado na posição.

Parâmetros

<i>pos</i>	índice da posição escolhida (a partir de 1)
------------	---

Definição na linha 134 do arquivo `playlist.cpp`.

```
00134 {
00135     songs->removePosition(pos);
00136 }
```

4.5.3.17 saveAlltofile()

```
void Playlist::saveAlltofile (
    no * current,
    ofstream & myfile )
```

Função recursiva que adiciona uma playlist a um arquivo "Playlist.txt".

Parâmetros

<i>current</i>	recebe o nó do índice atual da música na playlist
<i>myfile</i>	recebe o arquivo que os dados serão inseridos.

Condição de parada

Exibe a música atual

Chamada recursiva para que todas as músicas sejam exibidas

Definição na linha 232 do arquivo [playlist.cpp](#).

```
00232                                     {
00234     if (current == nullptr) {
00235         count = 1;
00236         return;
00237     }
00238     tempStr = current->data.getTitle() + ":" + current->data.getArtist() + ",";
00239     if(current->next == nullptr){
00240         tempStr.erase(tempStr.size() -1 );
00241         myfile << tempStr;
00242     }
00243     else {
00244         myfile << tempStr;
00245     }
00246
00247     ++count;
00248     saveAlltofile(current->next, myfile);
00249 }
```

4.5.3.18 setName()

```
void Playlist::setName (
    string _name )
```

Essa função une a playlist atual com a playlist passada por referência (segPlaylist) e retorna uma nova playlist

Definição na linha 32 do arquivo [playlist.cpp](#).

4.5.4 Atributos

4.5.4.1 count

```
size_t Playlist::count [private]
```

Definição na linha 16 do arquivo [playlist.h](#).

4.5.4.2 name

```
string Playlist::name [private]
```

Definição na linha 13 do arquivo [playlist.h](#).

4.5.4.3 playing

```
no* Playlist::playing [private]
```

Definição na linha 15 do arquivo [playlist.h](#).

4.5.4.4 songs

```
LinkedList* Playlist::songs [private]
```

Definição na linha 14 do arquivo [playlist.h](#).

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

- [include/playlist.h](#)
- [src/playlist.cpp](#)

4.6 Referência da Classe Song

```
#include <musica.h>
```

Membros Públicos

- [Song](#) ()
Constrói um novo [Song::Song](#) object.
- [~Song](#) ()
Destroy um [Song::Song](#) object.
- string [getTitle](#) ()
Get que retorna o título de um "Song".
- void [setTitle](#) (string _title)
Set que define o título de um "Song".
- string [getArtist](#) ()
Get que retorna o artista de um "Song".
- void [setArtist](#) (string _artist)
Set que define o artista de um "Song".

Atributos Privados

- string [title](#)
- string [artist](#)

4.6.1 Descrição detalhada

Definição na linha 8 do arquivo [musica.h](#).

4.6.2 Construtores e Destrutores

4.6.2.1 Song()

```
Song::Song ( )
```

Constrói um novo `Song:: Song` object.

Definição na linha 16 do arquivo `musica.cpp`.

```
00016 {  
00017 }
```

4.6.2.2 ~Song()

```
Song::~~Song ( )
```

Destroy um `Song:: Song` object.

Definição na linha 22 do arquivo `musica.cpp`.

```
00022 {  
00023 }
```

4.6.3 Documentação das funções

4.6.3.1 getArtist()

```
string Song::getArtist ( )
```

Get que retorna o artista de um "Song".

Retorna

string Artista da música.

Definição na linha 37 do arquivo `musica.cpp`.

```
00037 {  
00038     return artist;  
00039 }
```

4.6.3.2 getTitle()

```
string Song::getTitle ( )
```

Get que retorna o título de um "Song".

Retorna

string título da música.

Definição na linha 29 do arquivo [musica.cpp](#).

```
00029 {  
00030     return title;  
00031 }
```

4.6.3.3 setArtist()

```
void Song::setArtist (  
    string _artist )
```

Set que define o artista de um "Song".

Parâmetros

<code>_artist</code>	
----------------------	--

Definição na linha 53 do arquivo [musica.cpp](#).

```
00053 {  
00054     artist = _artist;  
00055 }
```

4.6.3.4 setTitle()

```
void Song::setTitle (  
    string _title )
```

Set que define o título de um "Song".

Parâmetros

<code>_title</code>	
---------------------	--

Definição na linha 45 do arquivo [musica.cpp](#).

```
00045 {  
00046     title = _title;  
00047 }
```

4.6.4 Atributos

4.6.4.1 artist

```
string Song::artist [private]
```

Definição na linha 11 do arquivo [musica.h](#).

4.6.4.2 title

```
string Song::title [private]
```

Definição na linha 10 do arquivo [musica.h](#).

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

- include/[musica.h](#)
- src/[musica.cpp](#)

Chapter 5

Arquivos

5.1 Referência do Arquivo include/listaLigada.h

```
#include "musica.h"
#include <iostream>
```

Gráfico de dependência de inclusões para listaLigada.h: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- struct `no`
- class `LinkedList`

5.2 listaLigada.h

[Vá para a documentação desse arquivo.](#)

```
00001 #ifndef LISTALIGADA_H
00002 #define LISTALIGADA_H
00003
00004 #include "musica.h"
00005 #include <iostream>
00006 //O struct abaixo representa um nó da lista ligada.
00007 struct no {
00008     Song data; //Armazena um objeto do tipo música.
00009     no* next; //Ponteiro para o próximo nó.
00010 };
00011 //A classe abaixo representa uma lista ligada.
00012 class LinkedList {
00013     private:
00014         no* head; //Ponteiro para o inicio da lista.
00015         no* tail; //Ponteiro para o final da lista.z
00016         size_t size; //Tamanho da lista.
00017     public:
00018         LinkedList(); //Construtor da lista ligada.
00019         ~LinkedList(); //Destrutor da lista ligada.
00020
00021         LinkedList(const LinkedList& oldList); //Construtor cópia da lista ligada.
00022
00023         LinkedList operator+ (const LinkedList& secondList); //Concatenação de listas.
00024
00025         void operator» (no* & lastno); //Extraí o último elemento da lista.
00026         void operator« (no* & newno); //Insere um nó no fim da lista.
00027
00028         no* getHead(); //Obtém o ponteiro do head.
00029         no* getTail(); //Obtém o ponteiro do tail.
00030         size_t getSize(); //Obtém o tamanho da lista.
00031
```

```

00032 void insertStart(Song value); //Insere um nó no início.
00033 int insertEnd(Song value); //Insere um nó no fim.
00034 void insertPosition(size_t pos, Song value); //Insere um nó na posição específica.
00035
00036 void insertEnd(LinkedList& toInsert); //Sobrecarga do método de inserção.
00037
00038 void removeFirst(); //Elimina o primeiro nó.
00039 void removeLast(); //Elimina o último nó.
00040 void removePosition(size_t pos); //Elimina o nó na posição específica.
00041
00042 void removePosition(LinkedList& toRemove); //Sobrecarga do método de remoção.
00043
00044 no* search(Song searchSong); //Procura um nó usando os atributos da música.
00045 no* getno(size_t pos); //Retorna um nó com base na posição.
00046 size_t getPosition(Song searchSong); //Retorna a posição de uma música na lista.
00047
00048 void display(); //Exibe o conteúdo dos nós da lista.
00049 };
00050 #endif

```

5.3 Referência do Arquivo include/listaPlaylists.h

```

#include "playlist.h"
#include <iostream>

```

Gráfico de dependência de inclusões para listaPlaylists.h: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- struct `no_`
- class `ListOfPlaylists`

5.4 listaPlaylists.h

Vá para a documentação desse arquivo.

```

00001 #ifndef LISTOFPPLAYLISTS_H
00002 #define LISTOFPPLAYLISTS_H
00003
00004 #include "playlist.h"
00005 #include <iostream>
00006
00007 //O struct abaixo representa um nó da lista ligada.
00008 struct no_ {
00009     Playlist* data; //Armazena um objeto do tipo ponteiro de playlist.
00010     no_* next; //Ponteiro para o próximo nó.
00011 };
00012
00013 //A classe abaixo representa uma lista ligada de playlists
00014 class ListOfPlaylists {
00015     private:
00016         no_* head; //Ponteiro para o primeiro nó.
00017         no_* tail; //Ponteiro para o último nó.
00018         size_t size; //Tamanho da lista.
00019     public:
00020         ListOfPlaylists(); //Construtor da lista ligada.
00021         ~ListOfPlaylists(); //Destrutor da lista ligada.
00022
00023         size_t getSize(); //Adquire o tamanho da lista.
00024
00025         Playlist* getPlaylist(size_t pos); //Retorna o ponteiro para a playlist usando a posição na lista.
00026         Playlist* searchPlaylist(std::string searchName); //Retorna o ponteiro para a playlist usando o
nome da playlist.
00027
00028         void insertPlaylist(Playlist* value); //Insere um nó da playlist no fim da lista.
00029         void removePlaylist(size_t pos); //Elimina o nó da playlist de uma posição específica.
00030         void removeFromAll(Song target); //Elimina uma música de todas as playlists de uma vez.
00031         void display(); //Exibe o conteúdo de todos os nós da lista.
00032
00033 };
00034
00035
00036 #endif

```

5.5 Referência do Arquivo include/musica.h

```
#include <iostream>
```

Gráfico de dependência de inclusões para musica.h: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class [Song](#)

5.6 musica.h

[Vá para a documentação desse arquivo.](#)

```
00001 #ifndef MUSICA_H
00002 #define MUSICA_H
00003
00004 #include <iostream>
00005 using namespace std;
00006
00007 //A classe abaixo representa uma musica armazenada.
00008 class Song {
00009     private:
00010         string title; //Título da música.
00011         string artist; //Artista da música.
00012     public:
00013         Song(); //Construtor da música.
00014         ~Song(); //Destrutor da música.
00015
00016         string getTitle(); //Adquire o título da música.
00017         void setTitle(string _title); //Adiciona o título da música.
00018
00019         string getArtist(); //Adquire o artista da música.
00020         void setArtist(string _artist); //Adiciona o artista da música.
00021 };
00022
00023
00024 #endif
```

5.7 Referência do Arquivo include/playlist.h

```
#include "listaLigada.h"
```

```
#include "musica.h"
```

```
#include <iostream>
```

Gráfico de dependência de inclusões para playlist.h: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Componentes

- class [Playlist](#)

5.8 playlist.h

Vá para a documentação desse arquivo.

```
00001 #ifndef PLAYLIST_H
00002 #define PLAYLIST_H
00003
00004 #include "listaLigada.h"
00005 #include "musica.h"
00006 #include <iostream>
00007 using namespace std;
00008
00009
00010 //A classe abaixo representa uma playlist.
00011 class Playlist {
00012     private:
00013         string name; //Nome da playlist.
00014         LinkedList* songs; //Ponteiro para a lista que armazena as músicas.
00015         no* playing; //Ponteiro para o nó da música que está sendo reproduzida.
00016         size_t count; //Contador para o método de display.
00017     public:
00018         Playlist(); //Construtor da playlist.
00019         ~Playlist(); //Destrutor da playlist.
00020
00021         Playlist(const Playlist& old); //Construtor cópia da playlist.
00022
00023         Playlist operator+ (Playlist& secondPlaylist); //Fusão de playlists.
00024         Playlist operator+ (Song& toAdd); //Método que copia a playlist e adiciona uma música.
00025         Playlist operator- (Playlist& secondPlaylist); //Diferença entre duas playlists.
00026         Playlist operator- (Song& toRemove); //Método que copia a playlist e remove uma música.
00027         void operator« (Song* &lastSong); //Método que extrai a última música da playlist.
00028         void operator« (Song* &newSong); //Método que adiciona uma música no final da playlist.
00029
00030         LinkedList* getSongs(); //Adquire o ponteiro da lista de músicas.
00031
00032         string getName(); //Adquire o nome da playlist.
00033         void setName(string _name); //Adiciona o nome da playlist.
00034
00035         void insertSong(size_t pos, Song value); //Adiciona uma música na playlist.
00036         void removeSong(size_t pos); //Elimina uma música da playlist.
00037         void moveSong(size_t start, size_t end); //Move a música para outra posição na playlist.
00038
00039         void insertSong(Playlist& toInsert); //Sobrecarga do método de inserção.
00040         size_t removeSong(Playlist& toRemove); //Sobrecarga do método de remoção.
00041
00042         no* playNext(); //Retorna a próxima música a ser reproduzida.
00043
00044         void displayAllSongs(no* current); //Exibe todas as músicas que compoem a playlist.
00045         void saveAlltofile(no* current, ofstream& myfile); //Salva todas as músicas que existem em uma
playlist em um arquivo
00046         void displayOne(no* current, int pos); //Exibe uma música específica da playlist.
00047 };
00048
00049 #endif
```

5.9 Referência do Arquivo include/utilitarios.h

```
#include <iostream>
```

Gráfico de dependência de inclusões para utilitarios.h: Este grafo mostra quais arquivos estão direta ou indiretamente relacionados com esse arquivo:

Funções

- string [toLowerCase](#) (string s)
- string [checkInt](#) (string s)
- void [testeSobrecarga](#) ()

Função que testa os métodos sobrecarregados.

- void [helpPage](#) ()

Função que imprime para o usuário os comandos que o programa pode executar.

- void [telainicial](#) ()

Função que imprime a tela inicial.

- void [stopPlaying](#) (int &playing, size_t &index)

Função que para a reprodução de músicas.

5.9.1 Funções

5.9.1.1 checkInt()

```
string checkInt (
    string s )
```

Essa função recebe uma string e verifica se há um inteiro em seu conteúdo.

Parâmetros

s	a string a ser verificada
---	---------------------------

Retorna

s se houver um inteiro ou aux se não.

Definição na linha 36 do arquivo [utilitarios.cpp](#).

```
00036 {
00037     string aux = "-1";
00038     if (isdigit(s[0]) == true) {
00039         return s;
00040     } else {
00041         return aux;
00042     }
00043 }
```

5.9.1.2 helpPage()

```
void helpPage ( )
```

Função que imprime para o usuário os comandos que o programa pode executar.

Definição na linha 273 do arquivo [utilitarios.cpp](#).

```
00273 {
00274
00275     cout << "Comandos para gerenciamento de músicas: " << endl << endl;
00276
00277     cout << "add - Adicionar uma música" << endl;
00278     cout << "del - Remover uma música" << endl;
00279     cout << "list - Listar todas as músicas" << endl;
00280     cout << "search - Buscar uma música" << endl << endl;
00281
00282     cout << "Comandos para gerenciamento de playlists: " << endl << endl;
00283
00284     cout << "addp - Adicionar uma playlist" << endl;
00285     cout << "delp - Remover uma playlist" << endl;
00286     cout << "listp - Listar todas as playlists" << endl << endl;
00287
00288     cout << "Comandos para reproduzir músicas de playlists: " << endl;
00289
00290     cout << "playp - Comece a tocar uma playlist" << endl;
00291     cout << "playn - Toque a próxima música de uma playlist." << endl;
00292     cout << "playb - Volte uma música." << endl;
00293     cout << "plays - Pare a reprodução de músicas." << endl << endl;
00294
00295     cout << "Comandos para gerenciamento de músicas em playlists: " << endl << endl;
00296
00297     cout << "addmp - Adicionar música a uma playlist" << endl;
```

```

00298     cout << "delmp - Remover música de uma playlist" << endl;
00299     cout << "mmp - Mover música numa playlist" << endl;
00300     cout << "listmp - Listar músicas de uma playlist" << endl;
00301     cout << "copyp - Criar uma cópia de uma playlist existente." << endl;
00302     cout << "mergcp - Unir duas playlists em uma única playlist" << endl;
00303     cout << "mergcp- - Remover músicas da segunda playlist da primeira e retornar uma nova playlist" <<
endl << endl;
00304     cout << "savetf - Salvar uma playlist em um arquivo" << endl;
00305
00306     cout << "Comandos para teste:" << endl;
00307
00308     cout << "addtp - Adicionar músicas e playlists pré-definidas ao programa." << endl;
00309     cout << "otest - Teste de métodos sobrecarregados." << endl << endl;
00310
00311     cout << "quit - Encerrar o programa." << endl;
00312     cout << "_____ " << endl << endl;
00313 }

```

5.9.1.3 stopPlaying()

```

void stopPlaying (
    int & playing,
    size_t & index )

```

Função que para a reprodução de músicas.

Parâmetros

<i>playing</i>	Status da reprodução.
<i>index</i>	Índice da música que está sendo reproduzida atualmente.

Definição na linha 330 do arquivo [utilitarios.cpp](#).

```

00330     {
00331     if(playing == 0){
00332         cout << "Nada está tocando no momento..." << endl;
00333     }
00334     if(playing == 1){
00335         cout << "Parando a reprodução!" << endl;
00336         index = 0;
00337         playing = 0;
00338     }
00339 }

```

5.9.1.4 telaInicial()

```

void telaInicial ( )

```

Função que imprime a tela inicial.

Definição na linha 318 do arquivo [utilitarios.cpp](#).

```

00318     {
00319     cout << "          ----- Tela Inicial -----" << endl;
00320     cout << "Para uma lista de comandos digite 'help' " << endl << endl;
00321     cout << "Digite um comando: ";
00322 }

```

5.9.1.5 testeSobrecarga()

```
void testeSobrecarga ( )
```

Função que testa os métodos sobrecarregados.

Definição na linha 48 do arquivo `utilitarios.cpp`.

```
00048 {
00049     LinkedList* testList = new LinkedList;
00050     Song tempSong;
00051     LinkedList* lista1 = new LinkedList; //Criando uma lista ligada...
00052     // Inserindo músicas previamente a uma lista...
00053
00054     // Adicionando primeira música
00055     tempSong.setTitle("Shelter");
00056     tempSong.setArtist("Porter Robinson");
00057     lista1->insertEnd(tempSong);
00058     // Adicionando segunda música
00059     tempSong.setTitle("Never Gonna Give You Up");
00060     tempSong.setArtist("Rick Astley");
00061     lista1->insertEnd(tempSong);
00062     //Imprimindo a lista...
00063     cout << "Lista 1:" << endl;
00064     lista1->display();
00065     cout << endl << " " << endl << endl;
00066
00067     // Cria uma segunda lista utilizando o construtor cópia.
00068     LinkedList* lista2 = new LinkedList(*lista1);
00069
00070     // Adiciona mais músicas a segunda lista
00071     tempSong.setTitle("Live and Learn");
00072     tempSong.setArtist("Crush 40");
00073     lista2->insertEnd(tempSong);
00074
00075     tempSong.setTitle("Bone Dry");
00076     tempSong.setArtist("Tristan");
00077     lista2->insertEnd(tempSong);
00078
00079     tempSong.setTitle("Lua de Cristal");
00080     tempSong.setArtist("Xuxa Meneghel");
00081     lista2->insertEnd(tempSong);
00082
00083     cout << "Lista 2 criada a partir da lista 1 com mais músicas:" << endl;
00084     lista2->display();
00085     cout << endl << " " << endl << endl;
00086
00087     lista2->removePosition(*lista1); //Removendo da lista 2 as músicas da lista 1 por meio de
sobrecarga...
00088
00089     cout << "Lista 2 sem os elementos da lista 1:" << endl;
00090     lista2->display();
00091
00092     //Inserindo as músicas dessa lista à lista global por meio de sobrecarga...
00093     testList->insertEnd(*lista2);
00094     cout << endl << " " << endl << endl;
00095
00096     // Criando uma nova lista a partir da concatenação das duas listas já existentes...
00097     LinkedList* lista3 = new LinkedList(*lista1 + *lista2);
00098     cout << "Lista 3 criada usando a lista 1 + lista 2: " << endl;
00099     lista3->display();
00100     cout << endl << " " << endl << endl;
00101
00102     // Extraíndo o ultimo nó da lista 3...
00103     no* temp = new no;
00104     *lista3 >> temp;
00105
00106     cout << "Lista 3 pós-extração:" << endl;
00107     lista3->display();
00108     cout << endl << "Música extraída: ";
00109     cout << temp->data.getTitle() << " - " << temp->data.getArtist() << endl;
00110     cout << endl << " " << endl << endl;
00111
00112     // Retornando o nó final da lista 3...
00113     *lista3 << temp;
00114     cout << "Lista 3 após inserção:" << endl;
00115     lista3->display();
00116     cout << endl << " " << endl << endl;
00117
00118     // Inserindo músicas previamente a uma playlist
00119     Playlist* p1 = new Playlist;
00120     p1->setName("Playlist 1");
00121
00122     tempSong.setTitle("Never Gonna Give You Up");
```

```

00123     tempSong.setArtist("Rick Astley");
00124     p1->insertSong(1, tempSong);
00125
00126     tempSong.setTitle("Shelter");
00127     tempSong.setArtist("Porter Robinson");
00128     p1->insertSong(2, tempSong);
00129
00130     cout << p1->getName() << ":" << endl;
00131     p1->displayAllSongs(p1->getSongs()->getHead());
00132     cout << endl << " " << endl << endl;
00133     // Inserindo músicas em uma segunda playlist...
00134     Playlist* p2 = new Playlist;
00135     p2->setName("Playlist 2");
00136
00137     tempSong.setTitle("Running in the 90s");
00138     tempSong.setArtist("Max Coveri");
00139     p2->insertSong(1, tempSong);
00140
00141     tempSong.setTitle("It's My Life");
00142     tempSong.setArtist("No Doubt");
00143     p2->insertSong(2, tempSong);
00144
00145     cout << p2->getName() << ":" << endl;
00146     p2->displayAllSongs(p2->getSongs()->getHead());
00147     cout << endl;
00148     // Inserindo músicas de p1 em p2 usando sobrecarga...
00149     p2->insertSong(*p1);
00150     cout << p2->getName() << " com as músicas da playlist 1:" << endl;
00151     p2->displayAllSongs(p2->getSongs()->getHead());
00152     cout << endl << " " << endl << endl;
00153
00154     // Removendo de p2 as músicas de p1 usando sobrecarga...
00155     size_t count = p2->removeSong(*p1);
00156     cout << p2->getName() << " sem as " << count << " músicas da playlist 1:" << endl;
00157     p2->displayAllSongs(p2->getSongs()->getHead());
00158     cout << endl << " " << endl << endl;
00159
00160     // Criando uma playlist p3 idêntica a p1 utilizando o construtor cópia...
00161     Playlist* p3 = new Playlist(*p1);
00162     p3->setName("Playlist 3");
00163     cout << p3->getName() << " cópia de playlist 1:" << endl;
00164     p3->displayAllSongs(p3->getSongs()->getHead());
00165     cout << endl << " " << endl << endl;
00166
00167     // Adicionando mais músicas a p1...
00168     tempSong.setTitle("Love Sux");
00169     tempSong.setArtist("Avril Lavigne");
00170     p1->insertSong(3, tempSong);
00171
00172     tempSong.setTitle("Why do I");
00173     tempSong.setArtist("Set it Off");
00174     p1->insertSong(4, tempSong);
00175
00176     cout << p1->getName() << " com mais músicas:" << endl;
00177     p1->displayAllSongs(p1->getSongs()->getHead());
00178     cout << endl << " " << endl << endl;
00179
00180     // Criando uma playlist 4 resultante da união da playlist 1 e playlist 2...
00181     Playlist* p4 = new Playlist(*p1 + *p2);
00182     p4->setName("Playlist 4");
00183     cout << p4->getName() << " criada a partir da playlist 1 em união com a playlist 2:" << endl;
00184     p4->displayAllSongs(p4->getSongs()->getHead());
00185     cout << endl << " " << endl << endl;
00186
00187     // Criando uma playlist p5 usando a playlist 4 e o "tempSong"...
00188     tempSong.setTitle("Fooled Again");
00189     tempSong.setArtist("Lonely Bunker");
00190     Playlist* p5 = new Playlist(*p4 + tempSong);
00191     p5->setName("Playlist 5");
00192     cout << p5->getName() << " criada a partir da playlist 4 + uma nova música:" << endl;
00193     p5->displayAllSongs(p5->getSongs()->getHead());
00194     cout << endl << " " << endl << endl;
00195
00196     // Criando a playlist 6 baseando se na playlist 5 menos as músicas da playlist 3...
00197     Playlist* p6 = new Playlist(*p5 - *p3);
00198     p6->setName("Playlist 6");
00199     cout << p6->getName() << " criada da playlist 5 - playlist 3:" << endl;
00200     p6->displayAllSongs(p6->getSongs()->getHead());
00201     cout << endl << " " << endl << endl;
00202
00203     // Criando a playlist 7 a partir da playlist 6 menos o "tempSong"...
00204     Playlist* p7 = new Playlist(*p6 - tempSong);
00205     p7->setName("Playlist 7");
00206     cout << p7->getName() << " criada da playlist 6 - uma música:" << endl;
00207     p7->displayAllSongs(p7->getSongs()->getHead());
00208     cout << endl << " " << endl << endl;
00209

```

```

00210         // Extraíndo a última música da playlist 6...
00211         Song* songPtr = new Song;
00212         *p6 » songPtr;
00213
00214         cout << p6->getName() << " após extração:" << endl;
00215         p6->displayAllSongs(p6->getSongs()->getHead());
00216         cout << endl << "Música que foi extraída: ";
00217         cout << songPtr->getTitle() << " - " << songPtr->getArtist() << endl;
00218         cout << endl << "_____ " << endl << endl;
00219
00220         // Adicionando novamente a música ao fim da playlist 6...
00221         *p6 « songPtr;
00222         cout << p6->getName() << " após inserção:" << endl;
00223         p6->displayAllSongs(p6->getSongs()->getHead());
00224
00225         cout << endl << "_____ " << endl << endl;
00226         cout << "Eliminando variáveis..." << endl;
00227         //Deletando as listas e variáveis do teste...
00228         cout << "testList: ";
00229         delete testList;
00230         cout << "Concluído!" << endl;
00231         cout << "Lista 1: ";
00232         delete lista1;
00233         cout << "Concluído!" << endl;
00234         cout << "Lista 2: ";
00235         delete lista2;
00236         cout << "Concluído!" << endl;
00237         cout << "Lista 3: ";
00238         delete lista3;
00239         cout << "Concluído!" << endl;
00240         cout << "Playlist 1: ";
00241         delete p1;
00242         cout << "Concluído!" << endl;
00243         cout << "Playlist 2: ";
00244         delete p2;
00245         cout << "Concluído!" << endl;
00246         cout << "Playlist 3: ";
00247         delete p3;
00248         cout << "Concluído!" << endl;
00249         cout << "Playlist 4: ";
00250         delete p4;
00251         cout << "Concluído!" << endl;
00252         cout << "Playlist 5: ";
00253         delete p5;
00254         cout << "Concluído!" << endl;
00255         cout << "Playlist 6: ";
00256         delete p6;
00257         cout << "Concluído!" << endl;
00258         cout << "Playlist 7: ";
00259         delete p7;
00260         cout << "Concluído!" << endl;
00261         cout << "temp: ";
00262         delete temp;
00263         cout << "Concluído!" << endl;
00264         cout << "songPtr: ";
00265         delete songPtr;
00266         cout << "Concluído!" << endl << endl;
00267         cout << "Teste finalizado com sucesso!" << endl;
00268     }

```

5.9.1.6 toLowerCase()

```

string toLowerCase (
    string s )

```

Essa função recebe uma string, e percorre-a transformando todos os caracteres em minúsculos.

Parâmetros

s	é a string a ser transformada.
---	--------------------------------

Retorna

o resultado da conversão.

Definição na linha 23 do arquivo [utilitarios.cpp](#).

```
00023         {
00024     string result = "";
00025     for (size_t i = 0; i < s.size(); i++) {
00026         result += tolower(s[i]);
00027     }
00028     return result;
00029 }
```

5.10 utilitarios.h

Vá para a documentação desse arquivo.

```
00001 #ifndef UTILITARIOS_H
00002 #define UTILITARIOS_H
00003 using namespace std;
00004 #include <iostream>
00005
00006 string toLowercase(string s); //Converte uma string para minúsculo
00007 string checkInt(string s); //Adquire apenas o valor inteiro de uma string.
00008 void testeSobrecarga(); //Teste dos metodos sobrecarregados...
00009 void helpPage(); //Imprime a tela de ajuda.
00010 void telaInicial(); //Imprime o menu do programa.
00011 void stopPlaying(int& playing, size_t& index); //Para a reprodução de músicas.
00012 #endif
```

5.11 Referência do Arquivo README.md

5.12 Referência do Arquivo src/listaLigada.cpp

Funções necessárias para as listas ligadas de músicas.

```
#include <iostream>
#include "utilitarios.h"
#include "listaLigada.h"
#include "musica.h"
```

Gráfico de dependência de inclusões para listaLigada.cpp:

5.12.1 Descrição detalhada

Funções necessárias para as listas ligadas de músicas.

Autor

Erick Marques

Versão

0.1

Definição no arquivo [listaLigada.cpp](#).

5.13 listaLigada.cpp

Vá para a documentação desse arquivo.

```

00001
00008 #include <iostream>
00009 #include "utilitarios.h"
00010 #include "listaLigada.h"
00011 #include "musica.h"
00012 using namespace std;
00013
00015 LinkedList::LinkedList() {
00016     head = nullptr;
00017     tail = nullptr;
00018     size = 0;
00019 }
00020
00022 LinkedList::~LinkedList() {
00023     if (size != 0) {
00024         no* temp = nullptr;
00025         no* cur = nullptr;
00026
00027         cur = head;
00028
00029         while (cur != nullptr) {
00030             temp = cur->next;
00031             delete cur;
00032             cur = temp;
00033         }
00034     }
00035 }
00036
00037 //As funções abaixo retornam o head, tail e tamanho das listas, respectivamente.
00038 no* LinkedList::getHead() {
00039     return head;
00040 }
00041
00042 no* LinkedList::getTail() {
00043     return tail;
00044 }
00045
00046 size_t LinkedList::getSize() {
00047     return size;
00048 }
00049
00054 LinkedList::LinkedList(const LinkedList& oldList) {
00055     head = nullptr;
00056     tail = nullptr;
00057     size = 0;
00058     no* temp = oldList.head;
00059     while (temp != nullptr) {
00060         insertEnd(temp->data);
00061         temp = temp->next;
00062     }
00063 }
00069 LinkedList LinkedList::operator+ (const LinkedList& segLista) {
00070     LinkedList listaFinal;
00071
00072     //O loop abaixo percorre a lista atual (iniciando do head) e copia os valores para a lista final.
00073     no* temp = head;
00074     while (temp != nullptr) {
00075         listaFinal.insertEnd(temp->data);
00076
00077         temp = temp->next;
00078     }
00079     temp = segLista.head;
00080     while (temp != nullptr) {
00081         no* newno = new no;
00082
00083         newno->data = temp->data;
00084         newno->next = nullptr;
00085
00086         if (listaFinal.head == nullptr) {
00087             listaFinal.head = newno;
00088             listaFinal.tail = newno;
00089             newno = nullptr;
00090         } else {
00091             listaFinal.tail->next = newno;
00092             listaFinal.tail = newno;
00093         }
00094         listaFinal.size++;
00095         temp = temp->next;
00096     }
00097     return listaFinal; //Retornando a lista final.
00098 }
00099

```

```

00104 void LinkedList::operator» (no*& lastno){
00105     if (size > 0) {
00106         lastno->data = tail->data;
00107         lastno->next = tail->next;
00108         removeLast();
00109     } else {
00110         lastno = nullptr;
00111     }
00112 }
00113
00118 void LinkedList::operator« (no*& newno) {
00119     if (newno == nullptr) {
00120         return;
00121     } else {
00122         insertEnd(newno->data);
00123     }
00124 }
00125
00131 void LinkedList::insertStart(Song value) {
00132     //O If verifica e impede que a mesma música seja inserida novamente.
00133     if (search(value) != nullptr) {
00134         cout << "Esta música já foi adicionada!" << endl << endl;
00135     } else {
00136         //Caso a música não tenha sido inserida, ela é adicionada normalmente. :)
00137         no* temp = new no;
00138         temp->data = value;
00139         temp->next = head;
00140         head = temp;
00141         ++size; //No final do processo, o tamanho da lista é aumentado.
00142     }
00143 }
00144
00150 int LinkedList::insertEnd(Song value) {
00151     //O If verifica e impede que a mesma música seja inserida novamente.
00152     if (search(value) != nullptr) {
00153         cout << "Esta música já foi adicionada!" << endl << endl;
00154         return 0;
00155     } else {
00156         //Caso a música não tenha sido inserida, ela é adicionada normalmente. :)
00157         no* temp = new no;
00158         temp->data = value;
00159         temp->next = nullptr;
00160         if (head == nullptr) {
00161             head = temp;
00162             tail = temp;
00163             temp = nullptr;
00164         } else {
00165             tail->next = temp;
00166             tail = temp;
00167         }
00168         ++size; //No final do processo, o tamanho da lista é aumentado.
00169         return 1;
00170     }
00171 }
00172
00177 void LinkedList::insertEnd(LinkedList& toInsert) {
00178     //O if abaixo verifica se a nova lista não está vazia.
00179     if (toInsert.getSize() < 1) {
00180         return;
00181     } else {
00182         no* temp = toInsert.getHead();
00183         //Esse loop adiciona cada elemento no fim da lista atual
00184         while (temp != nullptr) {
00185             insertEnd(temp->data);
00186             temp = temp->next;
00187         }
00188     }
00189 }
00196 void LinkedList::insertPosition(size_t pos, Song value) {
00197     no* pre = nullptr;
00198     no* cur = nullptr;
00199     //Verifica se a posição escolhida é a primeira ou maior que a última.
00200     if (pos == 1) {
00201         if (size == 0) {
00202             insertEnd(value);
00203         } else {
00204             insertStart(value);
00205         }
00206     } else if (pos <= size) {
00207         //Esse if verifica e impede que a mesma música seja inserida novamente.
00208         if (search(value) != nullptr) {
00209             cout << "Esta música já foi adicionada!" << endl << endl;
00210         } else {
00211             no* temp = new no;
00212             cur = head;
00213             //Esse loop organiza a lista ao inserir a nova música.
00214             for (size_t i = 1; i < pos; ++i) {

```



```

00215         pre = cur;
00216         cur = cur->next;
00217     }
00218     temp->data = value;
00219     pre->next = temp;
00220     temp->next = cur;
00221
00222     ++size; //Aumentando o valor da lista após a adição da nova música.
00223 }
00224 } else {
00225     insertEnd(value);
00226 }
00227 }
00228
00230 void LinkedList::removeFirst() {
00231     no* temp = head;
00232     head = head->next;
00233     --size; //Reduzindo o tamanho da lista após remover o valor.
00234     delete temp; //Deletando o valor temporário.
00235 }
00236
00238 void LinkedList::removeLast() {
00239     no* pre = nullptr;
00240     no* cur = nullptr;
00241     cur = head;
00242
00243     while (cur->next != nullptr) {
00244         pre = cur;
00245         cur = cur->next;
00246     }
00247
00248     tail = pre;
00249     pre->next = nullptr;
00250
00251     --size; //Reduzindo o tamanho da lista após remover o valor.
00252     delete cur;
00253 }
00254
00260 void LinkedList::removePosition(size_t pos) {
00261     no* pre = nullptr;
00262     no* cur = nullptr;
00263     //Verifica se a posição escolhida é a primeira ou maior que a última.
00264     if (pos == 1) {
00265         removeFirst();
00266     } else if (pos < size) {
00267         cur = head;
00268
00269         for (size_t i = 1; i < pos; ++i) {
00270             pre = cur;
00271             cur = cur->next;
00272         }
00273
00274         pre->next = cur->next;
00275
00276         --size;
00277         delete cur;
00278     } else if (pos < 1) {
00279         return;
00280     } else {
00281         removeLast();
00282     }
00283 }
00284
00289 void LinkedList::removePosition(LinkedList& toRemove) {
00290     //O if abaixo verifica se a nova lista não está vazia.
00291     if (toRemove.getSize() < 1) {
00292         return;
00293     } else {
00294         no* temp = toRemove.getHead();
00295
00296         while (temp != nullptr) {
00297             // Obtém a posição da música na lista atual
00298             size_t pos = getPosition(temp->data);
00299
00300             // Remove da lista atual
00301             if (pos > 0) {
00302                 removePosition(pos);
00303             }
00304
00305             temp = temp->next;
00306         }
00307     }
00308 }
00309
00316 no* LinkedList::search(Song searchSong) {
00317     no* temp = head;
00318

```

```

00319 while (temp != nullptr) {
00320     Song s = temp->data;
00321
00322     /*Transforma os caracteres dos títulos e artistas em minúsculos (sem alterar os valores originais)
00323
00324     e os compara, em seguida retorna a posição correspondente ou zero caso não encontre.*/
00325     if ( toLowercase(s.getTitle()) == toLowercase(searchSong.getTitle()) &&
00326         toLowercase(s.getArtist()) == toLowercase(searchSong.getArtist()) ) {
00327         // Retorna o ponteiro para o nó correspondente
00328         return temp;
00329     }
00330     temp = temp->next;
00331 }
00332 // Caso não encontre, retorna nullptr
00333 return nullptr;
00334 }
00335
00341 no* LinkedList::getno(size_t pos) {
00342     if (pos < 1 || pos > size) {
00343         return nullptr; //Caso a posição seja inválida, o retorno é nullptr.
00344     } else {
00345         //Caso a posição seja válida, a lista é percorrida do head até o final.
00346         no* temp = head;
00347         for (size_t i = 1; i < pos; ++i) {
00348             temp = temp->next;
00349         }
00350         return temp; //Retornando o ponteiro para o nó correspondente.
00351     }
00352 }
00353
00359 size_t LinkedList::getPosition(Song searchSong) {
00360     no* temp = head;
00361     size_t pos = 1;
00362
00363     while (temp != nullptr) {
00364         Song s = temp->data;
00365         /*Transforma os caracteres dos títulos e artistas em minúsculos (sem alterar os valores
00366         originais)
00367         e os compara, em seguida retorna a posição correspondente ou zero caso não encontre.*/
00368         if ( toLowercase(s.getTitle()) == toLowercase(searchSong.getTitle()) &&
00369             toLowercase(s.getArtist()) == toLowercase(searchSong.getArtist()) ) {
00370             return pos;
00371         }
00372         temp = temp->next;
00373         ++pos;
00374     }
00375     return 0;
00376 }
00380 void LinkedList::display() {
00381     no* temp = head;
00382     size_t index = 1;
00383
00384     while (temp != nullptr) {
00385         Song s = temp->data;
00386         cout << index << " - " << s.getTitle() << " - " << s.getArtist() << endl;
00387         temp = temp->next;
00388         ++index;
00389     }
00390 }

```

5.14 Referência do Arquivo src/listaPlaylists.cpp

Funções necessárias para as listas ligadas de playlists.

```

#include <iostream>
#include "listaPlaylists.h"
#include "utilitarios.h"

```

Gráfico de dependência de inclusões para listaPlaylists.cpp:

5.14.1 Descrição detalhada

Funções necessárias para as listas ligadas de playlists.

Autor

Erick Marques

Versão

0.1

Definição no arquivo [listaPlaylists.cpp](#).

5.15 listaPlaylists.cpp

[Vá para a documentação desse arquivo.](#)

```
00001
00008 #include <iostream>
00009 #include "listaPlaylists.h"
00010 #include "utilitarios.h"
00011
00012 using namespace std;
00013
00015 ListOfPlaylists::ListOfPlaylists() {
00016     head = nullptr;
00017     tail = nullptr;
00018     size = 0;
00019 }
00021 ListOfPlaylists::~~ListOfPlaylists() {
00022     if (size != 0) {
00023         no_* temp = nullptr;
00024         no_* cur = nullptr;
00025         cur = head;
00026         while (cur != nullptr) {
00027             temp = cur->next;
00028             delete cur->data;
00029             delete cur;
00030             cur = temp;
00031         }
00032     }
00033 }
00035 size_t ListOfPlaylists::getSize() {
00036     return size;
00037 }
00043 Playlist* ListOfPlaylists::getPlaylist(size_t pos) {
00044     if (pos < 1 || pos > size) {
00045         return nullptr;
00046     } else {
00047         no_* temp = head;
00048         for (size_t i = 1; i < pos; ++i) {
00049             temp = temp->next;
00050         }
00051         return temp->data;
00052     }
00053 }
00055 }
00061 Playlist* ListOfPlaylists::searchPlaylist(string searchName) {
00062     no_* temp = head;
00063     while (temp != nullptr) {
00064         if ( toLowercase(temp->data->getName()) == toLowercase(searchName) ) {
00065             return temp->data;
00066         }
00067         temp = temp->next;
00068     }
00069     return nullptr;
00070 }
00071
00072 }
00077 void ListOfPlaylists::insertPlaylist(Playlist* value) {
00078     if (searchPlaylist(value->getName()) != nullptr) {
00079         cout << "Uma playlist com esse nome já existe!" << endl << endl;
00080     } else {
00081         no_* temp = new no_;
00082         temp->data = value;
00083         temp->next = nullptr;
00084         if (head == nullptr) {
00085             head = temp;
00086             tail = temp;
00087             temp = nullptr;
00088         } else {
00089             tail->next = temp;
00090             tail = temp;
00091         }
00092     }
00093 }
```

```

00091     }
00092     ++size;
00093     cout << endl << "Playlist criada com sucesso!" << endl;
00094 }
00095 }
00096
00101 void ListOfPlaylists::removePlaylist(size_t pos) {
00102     no_* pre = nullptr;
00103     no_* cur = nullptr;
00104
00105     cur = head;
00106     if (pos == 1) {
00107         head = head->next;
00108     } else if (pos < size) {
00109         for (size_t i = 1; i < pos; ++i) {
00110             pre = cur;
00111             cur = cur->next;
00112         }
00113         pre->next = cur->next;
00114     } else {
00115         while (cur->next != nullptr) {
00116             pre = cur;
00117             cur = cur->next;
00118         }
00119         tail = pre;
00120         pre->next = nullptr;
00121     }
00122 }
00123 --size; //Reduzindo o tamanho da lista após a operação.
00124 delete cur->data;
00125 delete cur;
00126 }
00127
00132 void ListOfPlaylists::removeFromAll(Song target) {
00133     no_* temp = head;
00134     //Loop que percorre todas as playlists armazenadas.
00135     while (temp != nullptr) {
00136         LinkedList* songs = temp->data->getSongs(); //Salvando a lista de músicas atual.
00137         //If que Verifica se a musica está na lista
00138         if (songs->search(target) != nullptr) {
00139             size_t pos = songs->getPosition(target); //Obtém o ID da música.
00140             songs->removePosition(pos); //elimina a música da lista.
00141         }
00142         temp = temp->next;
00143     }
00144 }
00148 void ListOfPlaylists::display() {
00149     no_* temp = head;
00150     size_t i = 1;
00151     string nome = "";
00152     while (temp != nullptr) {
00153         nome = temp->data->getName();
00154         cout << i << " - " << nome << endl;
00155         temp = temp->next;
00156         ++i;
00157     }
00158 }

```

5.16 Referência do Arquivo src/main.cpp

Projeto que organiza músicas e playlists em listas ligadas.

```

#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <iterator>
#include <algorithm>
#include "musica.h"
#include "listaLigada.h"
#include "utilitarios.h"
#include "playlist.h"
#include "listaPlaylists.h"

```

Gráfico de dependência de inclusões para main.cpp:

Funções

- int `main` ()

O menu principal do programa e todas as suas operações.

Variáveis

- string `chkint`
- int `playing` = 0

5.16.1 Descrição detalhada

Projeto que organiza músicas e playlists em listas ligadas.

Autor

Erick Marques

Versão

0.1

Definição no arquivo `main.cpp`.

5.16.2 Funções

5.16.2.1 `main()`

```
int main ( )
```

O menu principal do programa e todas as suas operações.

Definição na linha 29 do arquivo `main.cpp`.

```
00029 {
00030     // Cria a lista global de músicas do programa
00031     LinkedList* globalList = new LinkedList;
00032     // Cria a lista de armazenar as playlists do programa
00033     ListOfPlaylists* playlists = new ListOfPlaylists;
00034
00035
00036     no* searchResult = nullptr; // Ponteiro para o nó retornado na busca
00037     size_t index = 0; // Índice na lista para as interações com o usuário
00038     size_t index2 = 0; // Índice na lista para as interações com o usuário
00039     // Variáveis temporárias para armazenar as entradas do usuário
00040     Playlist* tempPlaylist = nullptr;
00041     Song tempSong;
00042     string tempTitle = "";
00043     string tempArtist = "";
00044     int option = 1; // Variável que mantém o programa em loop.
00045     // Executa o menu de funcionalidades enquanto a opção for diferente de 0
00046     while (option != 0) {
00047         int validcommand = 0;
00048         telaInicial();
00049         string chooser; //Declaração da string principal do programa.
00050         cin » chooser; //Lê a opção escolhida
```

```

00051     cout << endl << "_____ " << endl << endl;
00052     if(chooser == "add") { // Adicionar uma música
00053         validcommand = 1;
00054         // Le as entradas
00055         cout << "Título da música: ";
00056         cin.ignore(256, '\n'); //Ignorar o "Enter"
00057         getline(cin, tempTitle);
00058         cout << "Nome do artista: ";
00059         getline(cin, tempArtist);
00060
00061         // Atribui num objeto do tipo Song
00062         tempSong.setTitle(tempTitle);
00063         tempSong.setArtist(tempArtist);
00064
00065         // Adiciona ao final da lista global
00066         int endadded = globalList->insertEnd(tempSong);
00067         if (endadded == 1){
00068             cout << endl << "Música adicionada com sucesso!" << endl << endl;
00069         }
00070     }
00071     if(chooser == "del"){ // Remover uma música
00072         validcommand = 1;
00073         if (globalList->getSize() == 0) {
00074             cout << "Não existem músicas registradas..." << endl << endl;
00075         } else {
00076             // Imprime a lista de músicas com seus índices
00077             globalList->display();
00078
00079             // Lê a entrada
00080             cout << endl << "Insira o índice da música a ser removida (enumerada acima): ";
00081             cin >> chkint; //O input do usuário irá para chkint uma variável do tipo string.
00082             index = stoi(checkInt(chkint)); //Caso o valor em chkint seja de fato um inteiro, index
                                assumirá o valor dele.
00083             while (index < 1 || index > globalList->getSize()) {
00084                 cout << "Índice inválido! Tente novamente: ";
00085                 cin >> chkint;
00086                 index = stoi(checkInt(chkint));
00087             }
00088
00089             // Acessa a música pelo índice
00090             searchResult = globalList->getno(index);
00091             cout << "A música " << searchResult->data.getTitle() << " - " << searchResult->data.getArtist() <<
                                " foi deletada." << endl;
00092
00093             // Remove de todas as playlists
00094             if (playlists->getSize() > 0) {
00095                 playlists->removeFromAll(searchResult->data);
00096             }
00097
00098             // Remove da lista global
00099             globalList->removePosition(index);
00100         }
00101     }
00102     if(chooser == "list"){ // Listar todas as músicas
00103         validcommand = 1;
00104         // Processo para garantir que há músicas adicionadas
00105         if (globalList->getSize() == 0) {
00106             cout << "Ainda não há nada aqui. Experimente adicionar algumas músicas." << endl << endl;
00107         } else {
00108             cout << "Músicas armazenadas:" << endl << endl;
00109             // Imprime a lista de músicas com seus índices
00110             globalList->display();
00111             cout << endl;
00112         }
00113     }
00114     if(chooser == "play"){
00115         // Processo para garantir que há playlists adicionadas
00116         validcommand = 1;
00117         if (playlists->getSize() == 0) {
00118             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma!" << endl;
00119         } else {
00120             // Lê a entrada
00121             cout << "Playlists disponíveis: " << endl;
00122             playlists->display();
00123             cout << "Insira o índice da playlist desejada: ";
00124             cin >> chkint;
00125             index = stoi(checkInt(chkint));
00126             cout << endl << endl;
00127             // Processo para garantir que a entrada é válida
00128             while (index < 1 || index > playlists->getSize()) {
00129                 cout << "Índice inválido! Tente novamente: ";
00130                 cin >> chkint;
00131                 index = stoi(checkInt(chkint));
00132             }
00133
00134             // Obtém a playlist pelo índice
00135             tempPlaylist = playlists->getPlaylist(index);

```

```

00136
00137 // Processo para garantir que há músicas adicionadas
00138 if (tempPlaylist->getSongs()->getSize() == 0) {
00139     cout << "Ops, não há músicas nessa playlist, adicione algumas músicas e tente novamente! " <<
endl << endl;
00140 } else {
00141     // Imprime a lista de músicas com seus índices
00142     tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00143
00144     // Lê a entrada
00145     cout << "Insira o índice da música que será tocada (consulte a lista de músicas): ";
00146     cin >> chkint;
00147     index = stoi(checkInt(chkint));
00148
00149     // Processo para garantir que a entrada é válida
00150     while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00151         cout << "Índice inválido! Tente novamente: ";
00152         cin >> chkint;
00153         index = stoi(checkInt(chkint));
00154     }
00155     cout << endl << "Tocando agora: ";
00156     tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00157     playing = 1;
00158 }
00159 }
00160 }
00161 if(chooser == "playn"){
00162     // Processo para garantir que há playlists adicionadas
00163     validcommand = 1;
00164     if (playing == 0){
00165         cout << "Nada está tocando no momento..." << endl << endl;
00166     }
00167     if (playing == 1){
00168         if (playlists->getSize() == 0) {
00169             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00170         } else {
00171             index++;
00172         }
00173         if (index > tempPlaylist->getSongs()->getSize()) {
00174             cout << endl << "Sua playlist acabou, mas a música nunca acaba, recomeçando! :D" << endl <<
endl;
00175             index = 1;
00176         }
00177         cout << "Tocando agora: ";
00178         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00179         cout << endl << endl;
00180     }
00181 }
00182 if(chooser == "playb"){
00183     // Processo para garantir que há playlists adicionadas
00184     validcommand = 1;
00185     if (playing == 0){
00186         cout << "Nada está tocando no momento..." << endl << endl;
00187     }
00188     if (playing == 1){
00189         if (playlists->getSize() == 0) {
00190             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00191         } else {
00192             index--;
00193         }
00194         if (index < 1) {
00195             cout << endl << "Você está no início... Assim como o tempo, você não pode voltar atrás..." <<
endl << endl;
00196             index = 1;
00197         }
00198         cout << "Tocando agora: ";
00199         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00200         cout << endl << endl;
00201     }
00202 }
00203 if(chooser == "plays"){
00204     validcommand = 1;
00205     stopPlaying(playing, index);
00206 }
00207 if(chooser == "search") { // Buscar uma música
00208     validcommand = 1;
00209     cout << "Título da música: ";
00210     cin.ignore(256, '\n');
00211     getline(cin, tempTitle);
00212     cout << "Nome do artista: ";
00213     getline(cin, tempArtist);
00214
00215     // Atribui num objeto do tipo Song
00216     tempSong.setTitle(tempTitle);
00217     tempSong.setArtist(tempArtist);
00218
00219     // Efetua a busca

```

```

00220     searchResult = globalList->search(tempSong);
00221
00222     // Da o retorno
00223     if (searchResult == nullptr) {
00224         cout << endl << "Essa música não foi adicionada! Verifique se você digitou corretamente ou
adicione a música antes de tentar novamente." << endl << endl;
00225     } else {
00226         cout << endl << "A música " << searchResult->data.getTitle() << " - " <<
searchResult->data.getArtist() << " está armazenada!" << endl;
00227     }
00228     cout << endl;
00229 }
00230 if(chooser == "addp") { // Adicionar uma playlist
00231     validcommand = 1;
00232     cout << "Nome da playlist: ";
00233     cin.ignore(256, '\n');
00234     getline(cin, tempTitle);
00235
00236     // Aloca a nova playlist
00237     tempPlaylist = new Playlist;
00238
00239     // Adiciona o nome escolhido
00240     tempPlaylist->setName(tempTitle);
00241
00242     // Insere na lista
00243     playlists->insertPlaylist(tempPlaylist);
00244
00245     cout << endl;
00246 }
00247 if(chooser == "delp") { // Remover uma playlist
00248     // Processo para garantir que há playlists adicionadas
00249     validcommand = 1;
00250     if (playlists->getSize() == 0) {
00251         cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl <<
endl;
00252     } else {
00253         // Imprime a lista de playlists com seus índices
00254         cout << "Playlists registradas: " << endl;
00255         playlists->display();
00256
00257         // Lê a entrada
00258         cout << endl << "Insira o índice da playlist a ser removida: ";
00259         cin >> chkint;
00260         index = stoi(checkInt(chkint));
00261         // Processo para garantir que a entrada é válida
00262         while (index < 1 || index > playlists->getSize()) {
00263             cout << "Índice inválido! Tente novamente: ";
00264             cin >> chkint;
00265             index = stoi(checkInt(chkint));
00266         }
00267
00268         // Remove a playlist
00269         playlists->removePlaylist(index);
00270         cout << "Remoção concluída com sucesso." << endl << endl;
00271     }
00272 }
00273 if(chooser == "listp") { // Listar todas as playlists
00274     // Processo para garantir que as playlists foram adicionadas.
00275     validcommand = 1;
00276     if (playlists->getSize() == 0) {
00277         cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl <<
endl;
00278     } else {
00279         cout << "Playlists armazenadas atualmente:" << endl << endl;
00280         // Imprime a lista de músicas com seus índices
00281         playlists->display();
00282     }
00283 }
00284 if(chooser == "addmp"){ // Adicionar música a uma playlist
00285     // Processo para garantir que há playlists adicionadas
00286     validcommand = 1;
00287     if (playlists->getSize() == 0) {
00288         cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl <<
endl;
00289     } else {
00290         // Lê a entrada
00291         cout << "Playlists disponíveis: " << endl;
00292         playlists->display();
00293         cout << "Insira o índice da playlist desejada (enumerada acima): ";
00294         int validstuff = 0;
00295         int invalidstuff = 0;
00296         while (validstuff == 0){
00297             if(invalidstuff == 1){
00298                 cout << "Índice inválido! Tente Novamente: ";
00299             }
00300             cin >> chkint;
00301             if (isdigit(chkint[0]) == true) {

```



```

00302         index = stoi(chkint);
00303         validstuff = 1;
00304     } else {
00305         validstuff = 0;
00306         invalidstuff = 1;
00307     }
00308 }
00309 // Processo para garantir que a entrada é válida
00310 while (index < 1 || index > playlists->getSize()) {
00311     cout << endl << "Índice inválido! Tente novamente: ";
00312     cin >> chkint;
00313     index = stoi(checkInt(chkint));
00314 }
00315
00316 // Obtém a playlist pelo índice
00317 tempPlaylist = playlists->getPlaylist(index);
00318 cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;
00319
00320 // Le as entradas
00321 cout << "Titulo da música: ";
00322 cin.ignore(256, '\n');
00323 getline(cin, tempTitle);
00324 cout << "Nome do artista: ";
00325 getline(cin, tempArtist);
00326 cout << "Em qual posição deseja adicionar (verifique a lista de músicas da playlist): ";
00327 validstuff = 0;
00328 while (validstuff == 0) {
00329     cin >> chkint;
00330     if (isdigit(chkint[0]) == true) {
00331         index = stoi(chkint);
00332         validstuff = 1;
00333     } else {
00334         cout << "Índice inválido! Tente novamente: ";
00335         validstuff = 0;
00336     }
00337 }
00338
00339 // Processo para garantir que a posição é válida
00340 while (index < 1) {
00341     cout << endl << "Posição inválida. Tente novamente: ";
00342     cin >> chkint;
00343     index = stoi(checkInt(chkint));
00344 }
00345 // Atribui num objeto do tipo Song
00346 tempSong.setTitle(tempTitle);
00347 tempSong.setArtist(tempArtist);
00348
00349 // Busca pela música armazenada
00350 searchResult = globalList->search(tempSong);
00351 if (searchResult == nullptr) {
00352     cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
novamente! " << endl << endl;
00353 } else {
00354     // Adiciona na playlist
00355     tempPlaylist->insertSong(index, tempSong);
00356     cout << endl << "Música adicionada à playlist '" << tempPlaylist->getName() << "'" << endl <<
endl;
00357 }
00358 }
00359 }
00360 if(chooser == "delmp"){ // Remover música de uma playlist
00361     // Processo para garantir que há playlists adicionadas
00362     validcommand = 1;
00363     if (playlists->getSize() == 0) {
00364         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma!" << endl << endl;
00365     } else {
00366         // Lê a entrada
00367         cout << "Playlists disponíveis: " << endl;
00368         playlists->display();
00369         cout << "Insira o índice da playlist desejada: ";
00370         cin >> chkint;
00371         index = stoi(checkInt(chkint));
00372
00373         // Processo para garantir que a entrada é válida
00374         while (index < 1 || index > playlists->getSize()) {
00375             cout << endl << "Índice inválido! Tente novamente: ";
00376             cin >> chkint;
00377             index = stoi(checkInt(chkint));
00378         }
00379
00380         // Obtém a playlist pelo índice
00381         tempPlaylist = playlists->getPlaylist(index);
00382         cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;
00383
00384         // Processo para garantir que há músicas adicionadas
00385         if (tempPlaylist->getSongs()->getSize() == 0) {
00386             cout << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente

```

```

    novamente! " « endl « endl;
00387     } else {
00388         // Imprime a lista de músicas com seus índices
00389         tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00390
00391         // Lê a entrada
00392         cout << "Insira o índice da música a ser removida (consulte a lista de músicas): ";
00393         cin >> chkint;
00394         index = stoi(checkInt(chkint));
00395
00396         // Processo para garantir que a entrada é válida
00397         while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00398             cout << "Índice inválido! Tente novamente: ";
00399             cin >> chkint;
00400             index = stoi(checkInt(chkint));
00401         }
00402
00403         // Remove a música
00404         tempPlaylist->removeSong(index);
00405         cout << endl << "Música removida da playlist '" << tempPlaylist->getName() << "'" << endl « endl;
00406     }
00407 }
00408
00409 if(chooser == "mmp"){ // Mover música numa playlist
00410     // Processo para garantir que há playlists adicionadas
00411     validcommand = 1;
00412     if (playlists->getSize() == 0) {
00413         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " « endl;
00414     } else {
00415         // Lê a entrada
00416         cout << "Playlists disponíveis: " « endl;
00417         playlists->display();
00418         cout << "Insira o índice da playlist desejada: ";
00419         cin >> chkint;
00420         index = stoi(checkInt(chkint));
00421
00422         // Processo para garantir que a entrada é válida
00423         while (index < 1 || index > playlists->getSize()) {
00424             cout << endl << "Índice inválido! Tente novamente: ";
00425             cin >> chkint;
00426             index = stoi(checkInt(chkint));
00427         }
00428
00429         // Obtém a playlist pelo índice
00430         tempPlaylist = playlists->getPlaylist(index);
00431         cout << "Playlist selecionada: " « tempPlaylist->getName() « endl « endl;
00432
00433         // Processo para garantir que há músicas adicionadas
00434         if (tempPlaylist->getSongs()->getSize() == 0) {
00435             cout << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
novamente! " « endl « endl;
00436         } else {
00437             // Imprime a lista de músicas com seus índices
00438             cout << endl << "Músicas nesta playlist: " « endl;
00439             tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00440
00441             // Lê a entrada
00442             cout << endl << "Insira o índice da música a ser movida: ";
00443             cin >> chkint;
00444             index = stoi(checkInt(chkint));
00445
00446             // Processo para garantir que a entrada é válida
00447             while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00448                 cout << endl << "Índice inválido! Tente novamente: ";
00449                 cin >> chkint;
00450                 index = stoi(checkInt(chkint));
00451             }
00452
00453             // Lê a entrada
00454             cout << "Insira o índice da posição para qual deseja movê-la: ";
00455             cin >> chkint;
00456             index2 = stoi(checkInt(chkint));
00457
00458             // Processo para garantir que a entrada é válida
00459             while (index2 < 1 || index2 > tempPlaylist->getSongs()->getSize()) {
00460                 cout << endl << "Índice inválido! Tente novamente: ";
00461                 cin >> chkint;
00462                 index2 = stoi(checkInt(chkint));
00463             }
00464
00465             // Move a música
00466             tempPlaylist->moveSong(index, index2);
00467             cout << endl << "Posição alterada com sucesso." « endl « endl;
00468         }
00469     }
00470 }
00471 if(chooser == "listmp") { // Listar músicas de uma playlist

```

```

00472     // Processo para garantir que há playlists adicionadas
00473     validcommand = 1;
00474     if (playlists->getSize() == 0) {
00475         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00476     } else {
00477         // Lê a entrada
00478         cout << "Playlists disponíveis: " << endl;
00479         playlists->display();
00480         cout << "Insira o índice da playlist desejada: ";
00481         cin >> chkint;
00482         index = stoi(checkInt(chkint));
00483
00484         // Processo para garantir que a entrada é válida
00485         while (index < 1 || index > playlists->getSize()) {
00486             cout << "Índice inválido! Tente novamente: ";
00487             cin >> chkint;
00488             index = stoi(checkInt(chkint));
00489         }
00490
00491         // Obtém a playlist pelo índice
00492         tempPlaylist = playlists->getPlaylist(index);
00493
00494         // Processo para garantir que há músicas adicionadas
00495         if (tempPlaylist->getSongs()->getSize() == 0) {
00496             cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
00497             novamente! " << endl << endl;
00498         } else {
00499             cout << endl << endl << "Músicas da playlist '" << tempPlaylist->getName() << "': " << endl << endl;
00500             tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00501         }
00502         cout << endl;
00503     }
00504     if(chooser == "copy"){ //Função que cria uma playlist cópia de uma já existente.
00505         validcommand = 1;
00506         if (playlists->getSize() == 0) {
00507             cout << "Ops, nenhuma playlist foi adicionada até o momento, crie duas e tente novamente! " <<
00508             endl;
00509         } else {
00510             // Lê a entrada
00511             cout << "Playlists disponíveis: " << endl;
00512             playlists->display();
00513             cout << "Insira o índice da primeira playlist desejada: ";
00514             cin >> chkint;
00515             index = stoi(checkInt(chkint));
00516
00517             // Processo para garantir que a entrada é válida
00518             while (index < 1 || index > playlists->getSize()) {
00519                 cout << "Índice inválido! Tente novamente: ";
00520                 cin >> chkint;
00521                 index = stoi(checkInt(chkint));
00522             }
00523
00524             // Obtém a playlist pelo índice
00525             tempPlaylist = playlists->getPlaylist(index);
00526
00527             Playlist* tempPlaylistC = new Playlist(*tempPlaylist);
00528
00529             cout << endl << "Digite o nome da nova Playlist: ";
00530             cin.ignore(256, '\n');
00531             getline(cin, tempTitle);
00532
00533             // Adiciona o nome escolhido
00534             tempPlaylistC->setName(tempTitle);
00535
00536             // Insere na lista
00537             playlists->insertPlaylist(tempPlaylistC);
00538         }
00539         if (chooser == "mergep") { //Função que cria uma nova playlist que contém as músicas de duas
00540             playlists previamente criadas.
00541             validcommand = 1;
00542             if (playlists->getSize() == 0) {
00543                 cout << "Ops, nenhuma playlist foi adicionada até o momento, crie duas e tente novamente! " <<
00544                 endl;
00545             }
00546             if (playlists->getSize() == 1) {
00547                 cout << "Você precisa de pelo menos duas playlists para realizar essa operação, crie-as e tente
00548                 novamente! " << endl;
00549             } else {
00550                 // Lê a entrada
00551                 cout << "Playlists disponíveis: " << endl;
00552                 playlists->display();
00553                 cout << "Insira o índice da primeira playlist desejada: ";
00554                 cin >> chkint;
00555                 index = stoi(checkInt(chkint));

```

```

00554
00555 // Processo para garantir que a entrada é válida
00556 while (index < 1 || index > playlists->getSize()) {
00557     cout << "Índice inválido! Tente novamente: ";
00558     cin >> chkint;
00559     index = stoi(checkInt(chkint));
00560 }
00561
00562 // Obtém a playlist pelo índice
00563 tempPlaylist = playlists->getPlaylist(index);
00564
00565 Playlist* tempPlaylist2 = nullptr;
00566
00567 // Lê a entrada
00568 cout << "Playlists disponíveis: " << endl;
00569 playlists->display();
00570 cout << "Insira o índice da segunda playlist desejada: ";
00571 cin >> chkint;
00572 index2 = stoi(checkInt(chkint));
00573
00574 // Processo para garantir que a entrada é válida
00575 while (index2 < 1 || index2 > playlists->getSize() || index2 == index ) {
00576     cout << "Índice inválido! Tente novamente: ";
00577     cin >> chkint;
00578     index2 = stoi(checkInt(chkint));
00579 }
00580 tempPlaylist2 = playlists->getPlaylist(index2);
00581
00582 Playlist* tempPlaylist3 = new Playlist(*tempPlaylist + *tempPlaylist2); //p3 = p1+p2
00583
00584 cout << "Digite o nome da nova Playlist: ";
00585 cin.ignore(256, '\n');
00586 getline(cin, tempTitle);
00587
00588 // Adiciona o nome escolhido
00589 tempPlaylist3->setName(tempTitle);
00590
00591 // Insere na lista
00592 playlists->insertPlaylist(tempPlaylist3);
00593 }
00594 }
00595 if (chooser == "mergpe-"){ // Função que usando duas playlists previamente criadas cria uma nova
playlist com as músicas da segunda removidas da primeira.
00596     validcommand = 1;
00597     if (playlists->getSize() == 0) {
00598         cout << "Ops, nenhuma playlist foi adicionada até o momento, crie duas e tente novamente! " <<
endl;
00599     }
00600     if (playlists->getSize() == 1) {
00601         cout << "Você precisa de pelo menos duas playlists para realizar essa operação, crie-as e tente
novamente! " << endl;
00602     } else {
00603         // Lê a entrada
00604         cout << "Playlists disponíveis: " << endl;
00605         playlists->display();
00606         cout << "Insira o índice da primeira playlist desejada: ";
00607         cin >> chkint;
00608         index = stoi(checkInt(chkint));
00609
00610         // Processo para garantir que a entrada é válida
00611         while (index < 1 || index > playlists->getSize()) {
00612             cout << "Índice inválido! Tente novamente: ";
00613             cin >> chkint;
00614             index = stoi(checkInt(chkint));
00615         }
00616
00617         // Obtém a playlist pelo índice
00618         tempPlaylist = playlists->getPlaylist(index);
00619
00620         Playlist* tempPlaylist2 = nullptr;
00621
00622         // Lê a entrada
00623         cout << "Playlists disponíveis: " << endl;
00624         playlists->display();
00625         cout << "Insira o índice da segunda playlist desejada: ";
00626         cin >> chkint;
00627         index2 = stoi(checkInt(chkint));
00628
00629         // Processo para garantir que a entrada é válida
00630         while (index2 < 1 || index2 > playlists->getSize() || index2 == index ) {
00631             cout << "Índice inválido! Tente novamente: ";
00632             cin >> chkint;
00633             index2 = stoi(checkInt(chkint));
00634         }
00635         tempPlaylist2 = playlists->getPlaylist(index2);
00636
00637         Playlist* tempPlaylist3 = new Playlist(*tempPlaylist - *tempPlaylist2); //p3 = p1 - p2

```

```

00638
00639     cout << "Digite o nome da nova Playlist: ";
00640     cin.ignore(256, '\n');
00641     getline(cin, tempTitle);
00642
00643     // Adiciona o nome escolhido
00644     tempPlaylist3->setName(tempTitle);
00645
00646     // Insere na lista
00647     playlists->insertPlaylist(tempPlaylist3);
00648 }
00649
00650 if(chooser == "addtp"){ //Função que adiciona algumas músicas e playlists ao reprodutor
    automaticamente.
00651     validcommand = 1;
00652     // Adicionando primeira playlist.
00653     Playlist* Playlist_1 = new Playlist;
00654     Playlist_1->setName("Playlist 1");
00655     playlists->insertPlaylist(Playlist_1);
00656     // Adicionando segundo playlist.
00657     Playlist* Playlist_2 = new Playlist;
00658     Playlist_2->setName("Playlist 2");
00659     playlists->insertPlaylist(Playlist_2);
00660     // Adicionando primeira música
00661     tempSong.setTitle("Shelter");
00662     tempSong.setArtist("Porter Robinson");
00663     globalList->insertEnd(tempSong);
00664     //Adicionando a música na playlist 1
00665     Playlist_1->insertSong(1, tempSong);
00666     // Adicionando segunda música
00667     tempSong.setTitle("Never Gonna Give You Up");
00668     tempSong.setArtist("Rick Astley");
00669     globalList->insertEnd(tempSong);
00670     Playlist_1->insertSong(2, tempSong);
00671     Playlist_2->insertSong(1, tempSong);
00672     // Adicionando terceira música
00673     tempSong.setTitle("Firework");
00674     tempSong.setArtist("Katy Perry");
00675     globalList->insertEnd(tempSong);
00676     Playlist_1->insertSong(3, tempSong);
00677     Playlist_2->insertSong(2, tempSong);
00678     // Adicionando quarta música
00679     tempSong.setTitle("Complicated");
00680     tempSong.setArtist("Avril Lavigne");
00681     globalList->insertEnd(tempSong);
00682     Playlist_2->insertSong(3, tempSong);
00683     // Adicionando quinta música
00684     tempSong.setTitle("Jump");
00685     tempSong.setArtist("Van Haley");
00686     globalList->insertEnd(tempSong);
00687     Playlist_2->insertSong(4, tempSong);
00688     cout << "Músicas adicionadas com sucesso!" << endl << endl;
00689 }
00690 if (chooser == "help"){
00691     validcommand = 1;
00692     helpPage();
00693 }
00694 if (chooser == "savetf"){
00695     validcommand = 1;
00696     ofstream myfile;
00697
00698     cout << "Escolha a playlist que você quer armazenar: " << endl;
00699     playlists->display();
00700     cout << "Insira o índice da playlist desejada: ";
00701     cin >> chkint;
00702     index = stoi(checkInt(chkint));
00703     while (index < 1 || index > playlists->getSize()) {
00704         cout << "Índice inválido! Tente novamente: ";
00705         cin >> chkint;
00706         index = stoi(checkInt(chkint));
00707     }
00708     tempPlaylist = playlists->getPlaylist(index);
00709
00710     if (tempPlaylist->getSongs()->getSize() == 0) {
00711         cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
    novamente!" << endl << endl;
00712     } else {
00713         myfile.open("Playlists.txt", std::ios_base::app);
00714         // myfile << endl << endl << "Músicas da playlist '" << tempPlaylist->getName() << "':" << endl <<
    endl;
00715         myfile << tempPlaylist->getName() << ";";
00716         tempPlaylist->saveAlltofile(tempPlaylist->getSongs()->getHead(), myfile);
00717         myfile.seekp(-1, ios::end);
00718         myfile << endl;
00719         myfile.close();
00720     }
00721 }

```

```
00722     if (chooser == "otest"){ // Opção de teste para os métodos sobrecarregados
00723         validcommand = 1;
00724         testeSobrecarga();
00725         cout << "Pressione 'Enter' para continuar." << endl;
00726         getchar();
00727         getchar();
00728     }
00729     if (chooser == "quit"){ //Encerra o programa.
00730         validcommand = 1;
00731         cout << "Encerrando o programa." << endl << endl;
00732         option = 0;
00733         // Libera a memória das listas globais
00734         delete globalList;
00735         delete playlists;
00736         cout << "Programa Encerrado!" << endl;
00737     }
00738     if(validcommand == 0){
00739         cout << "Comando inválido!" << endl << endl;
00740     }
00741 }
00742 return 0;
00743 }
```

5.16.3 Variáveis

5.16.3.1 chkint

```
string chkint
```

Definição na linha 23 do arquivo [main.cpp](#).

5.16.3.2 playing

```
int playing = 0
```

Definição na linha 24 do arquivo [main.cpp](#).

5.17 main.cpp

Vá para a documentação desse arquivo.

```
00001
00008 #include <iostream>
00009 #include <cstdlib>
00010 #include <string>
00011 #include <fstream>
00012 #include <iterator>
00013 #include <algorithm>
00014
00015 #include "musica.h"
00016 #include "listaligada.h"
00017 #include "utilitarios.h"
00018 #include "playlist.h"
00019 #include "listaPlaylists.h"
00020
00021 using namespace std;
00022
00023 string chkint;
00024 int playing = 0;
00025
00029 int main() {
```

```

00030 // Cria a lista global de músicas do programa
00031 LinkedList* globalList = new LinkedList;
00032 // Cria a lista de armazenar as playlists do programa
00033 ListOfPlaylists* playlists = new ListOfPlaylists;
00034
00035
00036 no* searchResult = nullptr; // Ponteiro para o nó retornado na busca
00037 size_t index = 0; // Índice na lista para as interações com o usuário
00038 size_t index2 = 0; // Índice na lista para as interações com o usuário
00039 // Variáveis temporárias para armazenar as entradas do usuário
00040 Playlist* tempPlaylist = nullptr;
00041 Song tempSong;
00042 string tempTitle = "";
00043 string tempArtist = "";
00044 int option = 1; // Variável que mantém o programa em loop.
00045 // Executa o menu de funcionalidades enquanto a opção for diferente de 0
00046 while (option != 0) {
00047     int validcommand = 0;
00048     telaInicial();
00049     string chooser; //Declaração da string principal do programa.
00050     cin >> chooser; //Lê a opção escolhida
00051     cout << endl << "_____ " << endl << endl;
00052     if(chooser == "add") { // Adicionar uma música
00053         validcommand = 1;
00054         // Le as entradas
00055         cout << "Título da música: ";
00056         cin.ignore(256, '\n'); //Ignorar o "Enter"
00057         getline(cin, tempTitle);
00058         cout << "Nome do artista: ";
00059         getline(cin, tempArtist);
00060
00061         // Atribui num objeto do tipo Song
00062         tempSong.setTitle(tempTitle);
00063         tempSong.setArtist(tempArtist);
00064
00065         // Adiciona ao final da lista global
00066         int endadded = globalList->insertEnd(tempSong);
00067         if (endadded == 1){
00068             cout << endl << "Música adicionada com sucesso!" << endl << endl;
00069         }
00070     }
00071     if(chooser == "del"){ // Remover uma música
00072         validcommand = 1;
00073         if (globalList->getSize() == 0) {
00074             cout << "Não existem músicas registradas..." << endl << endl;
00075         } else {
00076             // Imprime a lista de músicas com seus índices
00077             globalList->display();
00078
00079             // Lê a entrada
00080             cout << endl << "Insira o índice da música a ser removida (enumerada acima): ";
00081             cin >> chkint; //O input do usuário irá para chkint uma variável do tipo string.
00082             index = stoi(checkInt(chkint)); //Caso o valor em chkint seja de fato um inteiro, index
assumirá o valor dele.
00083             while (index < 1 || index > globalList->getSize()) {
00084                 cout << "Índice inválido! Tente novamente: ";
00085                 cin >> chkint;
00086                 index = stoi(checkInt(chkint));
00087             }
00088
00089             // Acessa a música pelo índice
00090             searchResult = globalList->getno(index);
00091             cout << "A música " << searchResult->data.getTitle() << " - " << searchResult->data.getArtist() <<
" foi deletada." << endl;
00092
00093             // Remove de todas as playlists
00094             if (playlists->getSize() > 0) {
00095                 playlists->removeFromAll(searchResult->data);
00096             }
00097
00098             // Remove da lista global
00099             globalList->removePosition(index);
00100         }
00101     }
00102     if(chooser == "list"){ // Listar todas as músicas
00103         validcommand = 1;
00104         // Processo para garantir que há músicas adicionadas
00105         if (globalList->getSize() == 0) {
00106             cout << "Ainda não há nada aqui. Experimente adicionar algumas músicas." << endl << endl;
00107         } else {
00108             cout << "Músicas armazenadas:" << endl << endl;
00109             // Imprime a lista de músicas com seus índices
00110             globalList->display();
00111             cout << endl;
00112         }
00113     }
00114     if(chooser == "playp"){

```

```

00115 // Processo para garantir que há playlists adicionadas
00116 validcommand = 1;
00117 if (playlists->getSize() == 0) {
00118     cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00119 } else {
00120     // Lê a entrada
00121     cout << "Playlists disponíveis: " << endl;
00122     playlists->display();
00123     cout << "Insira o índice da playlist desejada: ";
00124     cin >> chkint;
00125     index = stoi(checkInt(chkint));
00126     cout << endl << endl;
00127     // Processo para garantir que a entrada é válida
00128     while (index < 1 || index > playlists->getSize()) {
00129         cout << "Índice inválido! Tente novamente: ";
00130         cin >> chkint;
00131         index = stoi(checkInt(chkint));
00132     }
00133
00134     // Obtém a playlist pelo índice
00135     tempPlaylist = playlists->getPlaylist(index);
00136
00137     // Processo para garantir que há músicas adicionadas
00138     if (tempPlaylist->getSongs()->getSize() == 0) {
00139         cout << "Ops, não há músicas nessa playlist, adicione algumas músicas e tente novamente! " <<
endl << endl;
00140     } else {
00141         // Imprime a lista de músicas com seus índices
00142         tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00143
00144         // Lê a entrada
00145         cout << "Insira o índice da música que será tocada (consulte a lista de músicas): ";
00146         cin >> chkint;
00147         index = stoi(checkInt(chkint));
00148
00149         // Processo para garantir que a entrada é válida
00150         while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00151             cout << "Índice inválido! Tente novamente: ";
00152             cin >> chkint;
00153             index = stoi(checkInt(chkint));
00154         }
00155         cout << endl << "Tocando agora: ";
00156         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00157         playing = 1;
00158     }
00159 }
00160 }
00161 if(chooser == "playn"){
00162     // Processo para garantir que há playlists adicionadas
00163     validcommand = 1;
00164     if (playing == 0){
00165         cout << "Nada está tocando no momento..." << endl << endl;
00166     }
00167     if (playing == 1){
00168         if (playlists->getSize() == 0) {
00169             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00170         } else {
00171             index++;
00172         }
00173         if (index > tempPlaylist->getSongs()->getSize()) {
00174             cout << endl << "Sua playlist acabou, mas a música nunca acaba, recomeçando! :D" << endl <<
endl;
00175             index = 1;
00176         }
00177         cout << "Tocando agora: ";
00178         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00179         cout << endl << endl;
00180     }
00181 }
00182 if(chooser == "playb"){
00183     // Processo para garantir que há playlists adicionadas
00184     validcommand = 1;
00185     if (playing == 0){
00186         cout << "Nada está tocando no momento..." << endl << endl;
00187     }
00188     if (playing == 1){
00189         if (playlists->getSize() == 0) {
00190             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00191         } else {
00192             index--;
00193         }
00194         if (index < 1) {
00195             cout << endl << "Você está no início... Assim como o tempo, você não pode voltar atrás..." <<
endl << endl;
00196             index = 1;
00197         }
00198         cout << "Tocando agora: ";

```



```

00199         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00200         cout << endl << endl;
00201     }
00202 }
00203 if(chooser == "plays"){
00204     validcommand = 1;
00205     stopPlaying(playing, index);
00206 }
00207 if(chooser == "search") { // Buscar uma música
00208     validcommand = 1;
00209     cout << "Titulo da música: ";
00210     cin.ignore(256, '\n');
00211     getline(cin, tempTitle);
00212     cout << "Nome do artista: ";
00213     getline(cin, tempArtist);
00214
00215     // Atribui num objeto do tipo Song
00216     tempSong.setTitle(tempTitle);
00217     tempSong.setArtist(tempArtist);
00218
00219     // Efetua a busca
00220     searchResult = globalList->search(tempSong);
00221
00222     // Da o retorno
00223     if (searchResult == nullptr) {
00224         cout << endl << "Essa música não foi adicionada! Verifique se você digitou corretamente ou
adicione a música antes de tentar novamente." << endl << endl;
00225     } else {
00226         cout << endl << "A música " << searchResult->data.getTitle() << " - " <<
searchResult->data.getArtist() << " está armazenada!" << endl;
00227     }
00228     cout << endl;
00229 }
00230 if(chooser == "addp") { // Adicionar uma playlist
00231     validcommand = 1;
00232     cout << "Nome da playlist: ";
00233     cin.ignore(256, '\n');
00234     getline(cin, tempTitle);
00235
00236     // Aloca a nova playlist
00237     tempPlaylist = new Playlist;
00238
00239     // Adiciona o nome escolhido
00240     tempPlaylist->setName(tempTitle);
00241
00242     // Insere na lista
00243     playlists->insertPlaylist(tempPlaylist);
00244
00245     cout << endl;
00246 }
00247 if(chooser == "delp") { // Remover uma playlist
00248     // Processo para garantir que há playlists adicionadas
00249     validcommand = 1;
00250     if (playlists->getSize() == 0) {
00251         cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl <<
endl;
00252     } else {
00253         // Imprime a lista de playlists com seus indices
00254         cout << "Playlists registradas: " << endl;
00255         playlists->display();
00256
00257         // Lê a entrada
00258         cout << endl << "Insira o índice da playlist a ser removida: ";
00259         cin >> chkint;
00260         index = stoi(checkInt(chkint));
00261         // Processo para garantir que a entrada é válida
00262         while (index < 1 || index > playlists->getSize()) {
00263             cout << "Índice inválido! Tente novamente: ";
00264             cin >> chkint;
00265             index = stoi(checkInt(chkint));
00266         }
00267
00268         // Remove a playlist
00269         playlists->removePlaylist(index);
00270         cout << "Remoção concluída com sucesso." << endl << endl;
00271     }
00272 }
00273 if(chooser == "listp") { // Listar todas as playlists
00274     // Processo para garantir que as playlists foram adicionadas.
00275     validcommand = 1;
00276     if (playlists->getSize() == 0) {
00277         cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl <<
endl;
00278     } else {
00279         cout << "Playlists armazenadas atualmente:" << endl << endl;
00280         // Imprime a lista de músicas com seus indices
00281         playlists->display();

```

```

00282     }
00283 }
00284 if(chooser == "addmp"){ // Adicionar música a uma playlist
00285 // Processo para garantir que há playlists adicionadas
00286 validcommand = 1;
00287 if (playlists->getSize() == 0) {
00288     cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma!" << endl <<
endl;
00289 } else {
00290 // Lê a entrada
00291 cout << "Playlists disponíveis: " << endl;
00292 playlists->display();
00293 cout << "Insira o índice da playlist desejada (enumerada acima): ";
00294 int validstuff = 0;
00295 int invalidstuff = 0;
00296 while (validstuff == 0){
00297     if(invalidstuff == 1){
00298         cout << "Índice inválido! Tente Novamente: ";
00299     }
00300     cin >> chkint;
00301     if (isdigit(chkint[0]) == true) {
00302         index = stoi(chkint);
00303         validstuff = 1;
00304     } else {
00305         validstuff = 0;
00306         invalidstuff = 1;
00307     }
00308 }
00309 // Processo para garantir que a entrada é válida
00310 while (index < 1 || index > playlists->getSize()) {
00311     cout << endl << "Índice inválido! Tente novamente: ";
00312     cin >> chkint;
00313     index = stoi(checkInt(chkint));
00314 }
00315
00316 // Obtém a playlist pelo índice
00317 tempPlaylist = playlists->getPlaylist(index);
00318 cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;
00319
00320 // Le as entradas
00321 cout << "Título da música: ";
00322 cin.ignore(256, '\n');
00323 getline(cin, tempTitle);
00324 cout << "Nome do artista: ";
00325 getline(cin, tempArtist);
00326 cout << "Em qual posição deseja adicionar (verifique a lista de músicas da playlist): ";
00327 validstuff = 0;
00328 while (validstuff == 0){
00329     cin >> chkint;
00330     if (isdigit(chkint[0]) == true) {
00331         index = stoi(chkint);
00332         validstuff = 1;
00333     } else {
00334         cout << "Índice inválido! Tente novamente: ";
00335         validstuff = 0;
00336     }
00337 }
00338
00339 // Processo para garantir que a posição é válida
00340 while (index < 1) {
00341     cout << endl << "Posição inválida. Tente novamente: ";
00342     cin >> chkint;
00343     index = stoi(checkInt(chkint));
00344 }
00345 // Atribui num objeto do tipo Song
00346 tempSong.setTitle(tempTitle);
00347 tempSong.setArtist(tempArtist);
00348
00349 // Busca pela música armazenada
00350 searchResult = globalList->search(tempSong);
00351 if (searchResult == nullptr) {
00352     cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
novamente!" << endl << endl;
00353 } else {
00354 // Adiciona na playlist
00355 tempPlaylist->insertSong(index, tempSong);
00356 cout << endl << "Música adicionada à playlist '" << tempPlaylist->getName() << "'" << endl <<
endl;
00357 }
00358 }
00359 }
00360 if(chooser == "delmp"){ // Remover música de uma playlist
00361 // Processo para garantir que há playlists adicionadas
00362 validcommand = 1;
00363 if (playlists->getSize() == 0) {
00364     cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma!" << endl << endl;
00365 } else {

```

```

00366         // Lê a entrada
00367         cout << "Playlists disponíveis: " << endl;
00368         playlists->display();
00369         cout << "Insira o índice da playlist desejada: ";
00370         cin >> chkint;
00371         index = stoi(checkInt(chkint));
00372
00373         // Processo para garantir que a entrada é válida
00374         while (index < 1 || index > playlists->getSize()) {
00375             cout << endl << "Índice inválido! Tente novamente: ";
00376             cin >> chkint;
00377             index = stoi(checkInt(chkint));
00378         }
00379
00380         // Obtém a playlist pelo índice
00381         tempPlaylist = playlists->getPlaylist(index);
00382         cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;
00383
00384         // Processo para garantir que há músicas adicionadas
00385         if (tempPlaylist->getSongs()->getSize() == 0) {
00386             cout << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente novamente! " << endl << endl;
00387         } else {
00388             // Imprime a lista de músicas com seus índices
00389             tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00390
00391             // Lê a entrada
00392             cout << "Insira o índice da música a ser removida (consulte a lista de músicas): ";
00393             cin >> chkint;
00394             index = stoi(checkInt(chkint));
00395
00396             // Processo para garantir que a entrada é válida
00397             while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00398                 cout << "Índice inválido! Tente novamente: ";
00399                 cin >> chkint;
00400                 index = stoi(checkInt(chkint));
00401             }
00402
00403             // Remove a música
00404             tempPlaylist->removeSong(index);
00405             cout << endl << "Música removida da playlist '" << tempPlaylist->getName() << "' " << endl << endl;
00406         }
00407     }
00408 }
00409 if(chooser == "mmp"){ // Mover música numa playlist
00410     // Processo para garantir que há playlists adicionadas
00411     validcommand = 1;
00412     if (playlists->getSize() == 0) {
00413         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00414     } else {
00415         // Lê a entrada
00416         cout << "Playlists disponíveis: " << endl;
00417         playlists->display();
00418         cout << "Insira o índice da playlist desejada: ";
00419         cin >> chkint;
00420         index = stoi(checkInt(chkint));
00421
00422         // Processo para garantir que a entrada é válida
00423         while (index < 1 || index > playlists->getSize()) {
00424             cout << endl << "Índice inválido! Tente novamente: ";
00425             cin >> chkint;
00426             index = stoi(checkInt(chkint));
00427         }
00428
00429         // Obtém a playlist pelo índice
00430         tempPlaylist = playlists->getPlaylist(index);
00431         cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;
00432
00433         // Processo para garantir que há músicas adicionadas
00434         if (tempPlaylist->getSongs()->getSize() == 0) {
00435             cout << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente novamente! " << endl << endl;
00436         } else {
00437             // Imprime a lista de músicas com seus índices
00438             cout << endl << "Músicas nesta playlist: " << endl;
00439             tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00440
00441             // Lê a entrada
00442             cout << endl << "Insira o índice da música a ser movida: ";
00443             cin >> chkint;
00444             index = stoi(checkInt(chkint));
00445
00446             // Processo para garantir que a entrada é válida
00447             while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00448                 cout << endl << "Índice inválido! Tente novamente: ";
00449                 cin >> chkint;
00450                 index = stoi(checkInt(chkint));

```

```

00451     }
00452
00453     // Lê a entrada
00454     cout << "Insira o índice da posição para qual deseja movê-la: ";
00455     cin >> chkint;
00456     index2 = stoi(checkInt(chkint));
00457
00458     // Processo para garantir que a entrada é válida
00459     while (index2 < 1 || index2 > tempPlaylist->getSongs()->getSize()) {
00460         cout << endl << "Índice inválido! Tente novamente: ";
00461         cin >> chkint;
00462         index2 = stoi(checkInt(chkint));
00463     }
00464
00465     // Move a música
00466     tempPlaylist->moveSong(index, index2);
00467     cout << endl << "Posição alterada com sucesso." << endl << endl;
00468 }
00469 }
00470
00471 if(chooser == "listmp") { // Listar músicas de uma playlist
00472     // Processo para garantir que há playlists adicionadas
00473     validcommand = 1;
00474     if (playlists->getSize() == 0) {
00475         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00476     } else {
00477         // Lê a entrada
00478         cout << "Playlists disponíveis: " << endl;
00479         playlists->display();
00480         cout << "Insira o índice da playlist desejada: ";
00481         cin >> chkint;
00482         index = stoi(checkInt(chkint));
00483
00484         // Processo para garantir que a entrada é válida
00485         while (index < 1 || index > playlists->getSize()) {
00486             cout << "Índice inválido! Tente novamente: ";
00487             cin >> chkint;
00488             index = stoi(checkInt(chkint));
00489         }
00490
00491         // Obtém a playlist pelo índice
00492         tempPlaylist = playlists->getPlaylist(index);
00493
00494         // Processo para garantir que há músicas adicionadas
00495         if (tempPlaylist->getSongs()->getSize() == 0) {
00496             cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente novamente! " << endl << endl;
00497         } else {
00498             cout << endl << endl << "Músicas da playlist '" << tempPlaylist->getName() << "': " << endl << endl;
00499             tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00500         }
00501         cout << endl;
00502     }
00503 }
00504 if(chooser == "coppyp"){ //Função que cria uma playlist cópia de uma já existente.
00505     validcommand = 1;
00506     if (playlists->getSize() == 0) {
00507         cout << "Ops, nenhuma playlist foi adicionada até o momento, crie duas e tente novamente! " << endl;
00508     } else {
00509         // Lê a entrada
00510         cout << "Playlists disponíveis: " << endl;
00511         playlists->display();
00512         cout << "Insira o índice da primeira playlist desejada: ";
00513         cin >> chkint;
00514         index = stoi(checkInt(chkint));
00515
00516         // Processo para garantir que a entrada é válida
00517         while (index < 1 || index > playlists->getSize()) {
00518             cout << "Índice inválido! Tente novamente: ";
00519             cin >> chkint;
00520             index = stoi(checkInt(chkint));
00521         }
00522
00523         // Obtém a playlist pelo índice
00524         tempPlaylist = playlists->getPlaylist(index);
00525
00526         Playlist* tempPlaylistC = new Playlist(*tempPlaylist);
00527
00528         cout << endl << "Digite o nome da nova Playlist: ";
00529         cin.ignore(256, '\n');
00530         getline(cin, tempTitle);
00531
00532         // Adiciona o nome escolhido
00533         tempPlaylistC->setName(tempTitle);
00534
00535         // Insere na lista

```

```

00536         playlists->insertPlaylist(tempPlaylistC);
00537     }
00538 }
00539 if (chooser == "mergеп") { //Função que cria uma nova playlist que contém as músicas de duas
    playlists previamente criadas.
00540     validcommand = 1;
00541
00542     if (playlists->getSize() == 0) {
00543         cout << "Ops, nenhuma playlist foi adicionada até o momento, crie duas e tente novamente! " <<
endl;
00544     }
00545     if (playlists->getSize() == 1) {
00546         cout << "Você precisa de pelo menos duas playlists para realizar essa operação, crie-as e tente
novamente! " << endl;
00547     } else {
00548         // Lê a entrada
00549         cout << "Playlists disponíveis: " << endl;
00550         playlists->display();
00551         cout << "Insira o índice da primeira playlist desejada: ";
00552         cin >> chkint;
00553         index = stoi(checkInt(chkint));
00554
00555         // Processo para garantir que a entrada é válida
00556         while (index < 1 || index > playlists->getSize()) {
00557             cout << "Índice inválido! Tente novamente: ";
00558             cin >> chkint;
00559             index = stoi(checkInt(chkint));
00560         }
00561
00562         // Obtém a playlist pelo índice
00563         tempPlaylist = playlists->getPlaylist(index);
00564
00565         Playlist* tempPlaylist2 = nullptr;
00566
00567         // Lê a entrada
00568         cout << "Playlists disponíveis: " << endl;
00569         playlists->display();
00570         cout << "Insira o índice da segunda playlist desejada: ";
00571         cin >> chkint;
00572         index2 = stoi(checkInt(chkint));
00573
00574         // Processo para garantir que a entrada é válida
00575         while (index2 < 1 || index2 > playlists->getSize() || index2 == index) {
00576             cout << "Índice inválido! Tente novamente: ";
00577             cin >> chkint;
00578             index2 = stoi(checkInt(chkint));
00579         }
00580         tempPlaylist2 = playlists->getPlaylist(index2);
00581
00582         Playlist* tempPlaylist3 = new Playlist(*tempPlaylist + *tempPlaylist2); //p3 = p1+p2
00583
00584         cout << "Digite o nome da nova Playlist: ";
00585         cin.ignore(256, '\n');
00586         getline(cin, tempTitle);
00587
00588         // Adiciona o nome escolhido
00589         tempPlaylist3->setName(tempTitle);
00590
00591         // Insere na lista
00592         playlists->insertPlaylist(tempPlaylist3);
00593     }
00594 }
00595 if (chooser == "mergеп"){ // Função que usando duas playlists previamente criadas cria uma nova
playlist com as músicas da segunda removidas da primeira.
00596     validcommand = 1;
00597     if (playlists->getSize() == 0) {
00598         cout << "Ops, nenhuma playlist foi adicionada até o momento, crie duas e tente novamente! " <<
endl;
00599     }
00600     if (playlists->getSize() == 1) {
00601         cout << "Você precisa de pelo menos duas playlists para realizar essa operação, crie-as e tente
novamente! " << endl;
00602     } else {
00603         // Lê a entrada
00604         cout << "Playlists disponíveis: " << endl;
00605         playlists->display();
00606         cout << "Insira o índice da primeira playlist desejada: ";
00607         cin >> chkint;
00608         index = stoi(checkInt(chkint));
00609
00610         // Processo para garantir que a entrada é válida
00611         while (index < 1 || index > playlists->getSize()) {
00612             cout << "Índice inválido! Tente novamente: ";
00613             cin >> chkint;
00614             index = stoi(checkInt(chkint));
00615         }
00616

```

```

00617         // Obtém a playlist pelo índice
00618         tempPlaylist = playlists->getPlaylist(index);
00619
00620         Playlist* tempPlaylist2 = nullptr;
00621
00622         // Lê a entrada
00623         cout << "Playlists disponíveis: " << endl;
00624         playlists->display();
00625         cout << "Insira o índice da segunda playlist desejada: ";
00626         cin >> chkint;
00627         index2 = stoi(checkInt(chkint));
00628
00629         // Processo para garantir que a entrada é válida
00630         while (index2 < 1 || index2 > playlists->getSize() || index2 == index ) {
00631             cout << "Índice inválido! Tente novamente: ";
00632             cin >> chkint;
00633             index2 = stoi(checkInt(chkint));
00634         }
00635         tempPlaylist2 = playlists->getPlaylist(index2);
00636
00637         Playlist* tempPlaylist3 = new Playlist(*tempPlaylist - *tempPlaylist2); //p3 = p1 - p2
00638
00639         cout << "Digite o nome da nova Playlist: ";
00640         cin.ignore(256, '\n');
00641         getline(cin, tempTitle);
00642
00643         // Adiciona o nome escolhido
00644         tempPlaylist3->setName(tempTitle);
00645
00646         // Insere na lista
00647         playlists->insertPlaylist(tempPlaylist3);
00648     }
00649 }
00650 if(chooser == "addtp"){ //Função que adiciona algumas músicas e playlists ao reprodutor
    automaticamente.
00651     validcommand = 1;
00652     // Adicionando primeira playlist.
00653     Playlist* Playlist_1 = new Playlist;
00654     Playlist_1->setName("Playlist 1");
00655     playlists->insertPlaylist(Playlist_1);
00656     // Adicionando segundo playlist.
00657     Playlist* Playlist_2 = new Playlist;
00658     Playlist_2->setName("Playlist 2");
00659     playlists->insertPlaylist(Playlist_2);
00660     // Adicionando primeira música
00661     tempSong.setTitle("Shelter");
00662     tempSong.setArtist("Porter Robinson");
00663     globalList->insertEnd(tempSong);
00664     //Adicionando a música na playlist 1
00665     Playlist_1->insertSong(1, tempSong);
00666     // Adicionando segunda música
00667     tempSong.setTitle("Never Gonna Give You Up");
00668     tempSong.setArtist("Rick Astley");
00669     globalList->insertEnd(tempSong);
00670     Playlist_1->insertSong(2, tempSong);
00671     Playlist_2->insertSong(1, tempSong);
00672     // Adicionando terceira música
00673     tempSong.setTitle("Firework");
00674     tempSong.setArtist("Katy Perry");
00675     globalList->insertEnd(tempSong);
00676     Playlist_1->insertSong(3, tempSong);
00677     Playlist_2->insertSong(2, tempSong);
00678     // Adicionando quarta música
00679     tempSong.setTitle("Complicated");
00680     tempSong.setArtist("Avril Lavigne");
00681     globalList->insertEnd(tempSong);
00682     Playlist_2->insertSong(3, tempSong);
00683     // Adicionando quinta música
00684     tempSong.setTitle("Jump");
00685     tempSong.setArtist("Van Haley");
00686     globalList->insertEnd(tempSong);
00687     Playlist_2->insertSong(4, tempSong);
00688     cout << "Músicas adicionadas com sucesso!" << endl << endl;
00689 }
00690 if (chooser == "help"){
00691     validcommand = 1;
00692     helpPage();
00693 }
00694 if (chooser == "savetf"){
00695     validcommand = 1;
00696     ofstream myfile;
00697
00698     cout << "Escolha a playlist que você quer armazenar: " << endl;
00699     playlists->display();
00700     cout << "Insira o índice da playlist desejada: ";
00701     cin >> chkint;
00702     index = stoi(checkInt(chkint));

```

```

00703     while (index < 1 || index > playlists->getSize()) {
00704         cout << "Índice inválido! Tente novamente: ";
00705         cin >> chkint;
00706         index = stoi(checkInt(chkint));
00707     }
00708     tempPlaylist = playlists->getPlaylist(index);
00709
00710     if (tempPlaylist->getSongs()->getSize() == 0) {
00711         cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
novamente!" << endl << endl;
00712     } else {
00713         myfile.open("Playlists.txt", std::ios_base::app);
00714         // myfile << endl << endl << "Músicas da playlist '" << tempPlaylist->getName() << "': " << endl <<
endl;
00715         myfile << tempPlaylist->getName() << ";";
00716         tempPlaylist->saveAlltofile(tempPlaylist->getSongs()->getHead(), myfile);
00717         myfile.seekp(-1, ios::end);
00718         myfile << endl;
00719         myfile.close();
00720     }
00721 }
00722 if (chooser == "otest"){ // Opção de teste para os métodos sobrecarregados
00723     validcommand = 1;
00724     testeSobrecarga();
00725     cout << "Pressione 'Enter' para continuar." << endl;
00726     getchar();
00727     getchar();
00728 }
00729 if (chooser == "quit"){ //Encerra o programa.
00730     validcommand = 1;
00731     cout << "Encerrando o programa." << endl << endl;
00732     option = 0;
00733     // Libera a memória das listas globais
00734     delete globalList;
00735     delete playlists;
00736     cout << "Programa Encerrado!" << endl;
00737 }
00738 if(validcommand == 0){
00739     cout << "Comando inválido!" << endl << endl;
00740 }
00741 }
00742 return 0;
00743 }

```

5.18 Referência do Arquivo src/musica.cpp

Funções que definem e exibem informações das músicas.

```
#include "musica.h"
```

Gráfico de dependência de inclusões para musica.cpp:

5.18.1 Descrição detalhada

Funções que definem e exibem informações das músicas.

Autor

Erick Marques

Versão

0.1

Definição no arquivo [musica.cpp](#).

5.19 musica.cpp

Vá para a documentação desse arquivo.

```
00001
00008 #include "musica.h"
00009
00010 using namespace std;
00011
00016 Song::Song() {
00017 }
00022 Song::~Song() {
00023 }
00029 string Song::getTitle() {
00030     return title;
00031 }
00037 string Song::getArtist() {
00038     return artist;
00039 }
00045 void Song::setTitle(string _title) {
00046     title = _title;
00047 }
00053 void Song::setArtist(string _artist) {
00054     artist = _artist;
00055 }
```

5.20 Referência do Arquivo src/playlist.cpp

Funções que definem, exibem e permitem o funcionamento das playlists.

```
#include "playlist.h"
```

```
#include <fstream>
```

Gráfico de dependência de inclusões para playlist.cpp:

Variáveis

- string `tempStr`

5.20.1 Descrição detalhada

Funções que definem, exibem e permitem o funcionamento das playlists.

Autor

Erick Marques

Versão

0.1

Definição no arquivo `playlist.cpp`.

5.20.2 Variáveis

5.20.2.1 tempStr

string tempStr

Definição na linha 224 do arquivo [playlist.cpp](#).

5.21 playlist.cpp

Vá para a documentação desse arquivo.

```

00001
00008 #include "playlist.h"
00009 #include <fstream>
00010
00011 using namespace std;
00012
00014 Playlist::Playlist() {
00015     songs = new LinkedList;
00016     playing = nullptr;
00017     count = 1;
00018 }
00020 Playlist::~~Playlist() {
00021     delete songs;
00022 }
00024 LinkedList* Playlist::getSongs() {
00025     return songs;
00026 }
00028 string Playlist::getName() {
00029     return name;
00030 }
00032 void Playlist::setName(string _name) {
00033     name = _name;
00034 }
00036 Playlist Playlist::operator+ (Playlist& segPlaylist) {
00037     Playlist resultante;
00038     resultante.insertSong(*this);
00039     resultante.insertSong(segPlaylist);
00040     return resultante;
00041 }
00043 Playlist::Playlist(const Playlist& old) {
00044     songs = new LinkedList;
00045     playing = nullptr;
00046     count = 1;
00047     name = old.name;
00048     no* temp = old.songs->getHead();
00049     while (temp != nullptr) {
00050         songs->insertEnd(temp->data);
00051         temp = temp->next;
00052     }
00053 }
00055 Playlist Playlist::operator+ (Song& toAdd) {
00056     Playlist resultante;
00057     resultante.insertSong(*this);
00058     resultante.songs->insertEnd(toAdd);
00059     return resultante;
00060 }
00061
00067 Playlist Playlist::operator- (Playlist& segPlaylist) {
00068     Playlist resultante(*this);
00069     no* temp = segPlaylist.getSongs()->getHead();
00071     while (temp != nullptr) {
00073         size_t pos = resultante.getSongs()->getPosition(temp->data);
00074         if (pos > 0) {
00075             resultante.removeSong(pos);
00076         }
00077         temp = temp->next;
00078     }
00079     return resultante;
00080 }
00086 Playlist Playlist::operator- (Song& toRemove) {
00087     Playlist resultante(*this);
00088     size_t pos = resultante.getSongs()->getPosition(toRemove);
00090     if (pos > 0) {
00091         resultante.removeSong(pos);
00092     }
00093     return resultante;
00094 }
00099 void Playlist::operator» (Song*& lastSong) {
00100     if (songs->getSize() > 0) {

```

```

00101     no* temp = new no;
00102     *songs » temp;
00103     *lastSong = temp->data;
00104     delete temp;
00105 } else {
00106     lastSong = nullptr;
00107 }
00108 }
00113 void Playlist::operator« (Song*& newSong) {
00114     if (newSong == nullptr) {
00115         return;
00116     } else {
00117         songs->insertEnd(*newSong);
00118     }
00119 }
00125 void Playlist::insertSong(size_t pos, Song value) {
00126     songs->insertPosition(pos, value);
00127     playing = songs->getHead();
00128 }
00129
00134 void Playlist::removeSong(size_t pos) {
00135     songs->removePosition(pos);
00136 }
00137
00143 void Playlist::moveSong(size_t start, size_t end) {
00144     if (start != end) {
00146         no* target = songs->getno(start);
00147         Song value = target->data;
00149         songs->removePosition(start);
00151         if (end < start) {
00152             songs->insertPosition(end, value);
00153         } else {
00154             songs->insertPosition(end + 1, value);
00155         }
00156         playing = songs->getHead();
00157     }
00158 }
00159
00164 void Playlist::insertSong(Playlist& toInsert) {
00166     if (toInsert.getSongs()->getSize() < 1) {
00167         return;
00168     } else {
00169         songs->insertEnd(*toInsert.getSongs());
00170     }
00171 }
00172
00178 size_t Playlist::removeSong(Playlist& toRemove) {
00180     if (toRemove.getSongs()->getSize() < 1) {
00181         return 0;
00182     } else {
00183         size_t removed = 0;
00184         no* temp = toRemove.getSongs()->getHead();
00185         while (temp != nullptr) {
00186             size_t pos = getSongs()->getPosition(temp->data);
00188             if (pos > 0) {
00189                 removeSong(pos);
00190                 ++removed;
00191             }
00192             temp = temp->next;
00193         }
00194         return removed;
00195     }
00196 }
00197
00202 void Playlist::displayAllSongs(no* current) {
00204     if (current == nullptr) {
00205         count = 1;
00206         return;
00207     }
00208     cout « count « " - " « current->data.getTitle() « " - " « current->data.getArtist() « endl;
00209     ++count;
00210     displayAllSongs(current->next);
00211 }
00212
00213 /*void Playlist::saveAlltofile(no* current, ofstream& myfile) {
00214     // Condição de parada
00215     if (current == nullptr) {
00216         count = 1;
00217         return;
00218     }
00219     myfile « count « " - " « current->data.getTitle() « " - " « current->data.getArtist() « endl;
00220     //Exibe a música atual
00221     ++count;
00222     saveAlltofile(current->next, myfile); // Chamada recursiva para que todas as músicas sejam exibidas
00223 }*/
00224 string tempStr;

```

```

00225
00232 void Playlist::saveAlltofile(no* current, ofstream& myfile) {
00234     if (current == nullptr) {
00235         count = 1;
00236         return;
00237     }
00238     tempStr = current->data.getTitle() + ":" + current->data.getArtist() + ",";
00239     if(current->next == nullptr){
00240         tempStr.erase(tempStr.size() -1 );
00241         myfile << tempStr;
00242     }
00243     else {
00244         myfile << tempStr;
00245     }
00246
00247     ++count;
00248     saveAlltofile(current->next, myfile);
00249 }
00250
00251
00257 void Playlist::displayOne(no* current, int pos){
00258     if (current == nullptr) {
00259         count = 1;
00260         return;
00261     }
00262     if(count == pos){
00263         cout << current->data.getTitle() << " - " << current->data.getArtist() << endl;
00264     }
00265     ++count;
00266     displayOne(current->next, pos);
00267 }

```

5.22 Referência do Arquivo src/utilitarios.cpp

Função que transforma caracteres maiúsculos em minúsculos.

```

#include <iostream>
#include <cctype>
#include "utilitarios.h"
#include "musica.h"
#include "listaLigada.h"
#include "playlist.h"
#include "listaPlaylists.h"

```

Gráfico de dependência de inclusões para utilitarios.cpp:

Funções

- string [toLowerCase](#) (string s)
- string [checkInt](#) (string s)
- void [testeSobrecarga](#) ()
 - Função que testa os métodos sobrecarregados.*
- void [helpPage](#) ()
 - Função que imprime para o usuário os comandos que o programa pode executar.*
- void [telainicial](#) ()
 - Função que imprime a tela inicial.*
- void [stopPlaying](#) (int &playing, size_t &index)
 - Função que para a reprodução de músicas.*

5.22.1 Descrição detalhada

Função que transforma caracteres maiúsculos em minúsculos.

Autor

Erick Marques

Versão

0.1

Definição no arquivo [utilitarios.cpp](#).

5.22.2 Funções

5.22.2.1 checkInt()

```
string checkInt (
    string s )
```

Essa função recebe uma string e verifica se há um inteiro em seu conteúdo.

Parâmetros

s	a string a ser verificada
---	---------------------------

Retorna

s se houver um inteiro ou aux se não.

Definição na linha 36 do arquivo [utilitarios.cpp](#).

```
00036 {
00037     string aux = "-1";
00038     if (isdigit(s[0]) == true) {
00039         return s;
00040     } else {
00041         return aux;
00042     }
00043 }
```

5.22.2.2 helpPage()

```
void helpPage ( )
```

Função que imprime para o usuário os comandos que o programa pode executar.

Definição na linha 273 do arquivo [utilitarios.cpp](#).

```
00273     {
00274
00275     cout << "Comandos para gerenciamento de músicas: " << endl << endl;
00276
00277     cout << "add - Adicionar uma música" << endl;
00278     cout << "del - Remover uma música" << endl;
00279     cout << "list - Listar todas as músicas" << endl;
00280     cout << "search - Buscar uma música" << endl << endl;
00281
00282     cout << "Comandos para gerenciamento de playlists: " << endl << endl;
00283
00284     cout << "addp - Adicionar uma playlist" << endl;
00285     cout << "delp - Remover uma playlist" << endl;
00286     cout << "listp - Listar todas as playlists" << endl << endl;
00287
00288     cout << "Comandos para reproduzir músicas de playlists: " << endl;
00289
00290     cout << "playp - Comece a tocar uma playlist" << endl;
00291     cout << "playn - Toque a próxima música de uma playlist." << endl;
00292     cout << "playb - Volte uma música." << endl;
00293     cout << "plays - Pare a reprodução de músicas." << endl << endl;
00294
00295     cout << "Comandos para gerenciamento de músicas em playlists: " << endl << endl;
00296
00297     cout << "addmp - Adicionar música a uma playlist" << endl;
00298     cout << "delmp - Remover música de uma playlist" << endl;
00299     cout << "mmp - Mover música numa playlist" << endl;
00300     cout << "listmp - Listar músicas de uma playlist" << endl;
00301     cout << "copyp - Criar uma cópia de uma playlist existente." << endl;
00302     cout << "mergep - Unir duas playlists em uma única playlist" << endl;
00303     cout << "mergep- - Remover músicas da segunda playlist da primeira e retornar uma nova playlist" <<
endl << endl;
00304     cout << "savetf - Salvar uma playlist em um arquivo" << endl;
00305
00306     cout << "Comandos para teste:" << endl;
00307
00308     cout << "addtp - Adicionar músicas e playlists pré-definidas ao programa." << endl;
00309     cout << "otest - Teste de métodos sobrecarregados." << endl << endl;
00310
00311     cout << "quit - Encerrar o programa." << endl;
00312     cout << "_____ " << endl << endl;
00313 }
```

5.22.2.3 stopPlaying()

```
void stopPlaying (
    int & playing,
    size_t & index )
```

Função que para a reprodução de músicas.

Parâmetros

<i>playing</i>	Status da reprodução.
<i>index</i>	Índice da música que está sendo reproduzida atualmente.

Definição na linha 330 do arquivo [utilitarios.cpp](#).

```
00330     {
00331     if(playing == 0){
00332     cout << "Nada está tocando no momento..." << endl;
00333     }
00334     if(playing == 1){
00335     cout << "Parando a reprodução!" << endl;
00336     index = 0;
00337     playing = 0;
00338     }
00339 }
```

5.22.2.4 telaInicial()

```
void telaInicial ( )
```

Função que imprime a tela inicial.

Definição na linha 318 do arquivo [utilitarios.cpp](#).

```
00318     {
00319     cout << "      ----- Tela Inicial -----" << endl;
00320     cout << "Para uma lista de comandos digite 'help' " << endl << endl;
00321     cout << "Digite um comando: ";
00322 }
```

5.22.2.5 testeSobrecarga()

```
void testeSobrecarga ( )
```

Função que testa os métodos sobrecarregados.

Definição na linha 48 do arquivo [utilitarios.cpp](#).

```
00048     {
00049     LinkedList* testList = new LinkedList;
00050     Song tempSong;
00051     LinkedList* lista1 = new LinkedList; //Criando uma lista ligada...
00052     // Inserindo músicas previamente a uma lista...
00053
00054     // Adicionando primeira música
00055     tempSong.setTitle("Shelter");
00056     tempSong.setArtist("Porter Robinson");
00057     lista1->insertEnd(tempSong);
00058     // Adicionando segunda música
00059     tempSong.setTitle("Never Gonna Give You Up");
00060     tempSong.setArtist("Rick Astley");
00061     lista1->insertEnd(tempSong);
00062     //Imprimindo a lista...
00063     cout << "Lista 1:" << endl;
00064     lista1->display();
00065     cout << endl << "      " << endl << endl;
00066
00067     // Cria uma segunda lista utilizando o construtor cópia.
00068     LinkedList* lista2 = new LinkedList(*lista1);
00069
00070     // Adiciona mais músicas a segunda lista
00071     tempSong.setTitle("Live and Learn");
00072     tempSong.setArtist("Crush 40");
00073     lista2->insertEnd(tempSong);
00074
00075     tempSong.setTitle("Bone Dry");
00076     tempSong.setArtist("Tristan");
00077     lista2->insertEnd(tempSong);
00078
00079     tempSong.setTitle("Lua de Cristal");
00080     tempSong.setArtist("Xuxa Meneghel");
00081     lista2->insertEnd(tempSong);
00082
00083     cout << "Lista 2 criada a partir da lista 1 com mais músicas:" << endl;
00084     lista2->display();
00085     cout << endl << "      " << endl << endl;
00086
00087     lista2->removePosition(*lista1); //Removendo da lista 2 as músicas da lista 1 por meio de
sobrecarga...
00088
00089     cout << "Lista 2 sem os elementos da lista 1:" << endl;
00090     lista2->display();
00091
00092     //Inserindo as músicas dessa lista à lista global por meio de sobrecarga...
00093     testList->insertEnd(*lista2);
00094     cout << endl << "      " << endl << endl;
00095
00096     // Criando uma nova lista a partir da concatenação das duas listas já existentes...
00097     LinkedList* lista3 = new LinkedList(*lista1 + *lista2);
00098     cout << "Lista 3 criada usando a lista 1 + lista 2: " << endl;
00099     lista3->display();
00100     cout << endl << "      " << endl << endl;
```

```

00101
00102 // Extraíndo o ultimo nó da lista 3...
00103 no* temp = new no;
00104 *lista3 » temp;
00105
00106 cout << "Lista 3 pós-extração:" << endl;
00107 lista3->display();
00108 cout << endl << "Música extraída: ";
00109 cout << temp->data.getTitle() << " - " << temp->data.getArtist() << endl;
00110 cout << endl << "_____ " << endl << endl;
00111
00112 // Retornando o nó final da lista 3...
00113 *lista3 < temp;
00114 cout << "Lista 3 após inserção:" << endl;
00115 lista3->display();
00116 cout << endl << "_____ " << endl << endl;
00117
00118 // Inserindo músicas previamente a uma playlist
00119 Playlist* p1 = new Playlist;
00120 p1->setName("Playlist 1");
00121
00122 tempSong.setTitle("Never Gonna Give You Up");
00123 tempSong.setArtist("Rick Astley");
00124 p1->insertSong(1, tempSong);
00125
00126 tempSong.setTitle("Shelter");
00127 tempSong.setArtist("Porter Robinson");
00128 p1->insertSong(2, tempSong);
00129
00130 cout << p1->getName() << ":" << endl;
00131 p1->displayAllSongs(p1->getSongs()->getHead());
00132 cout << endl << "_____ " << endl << endl;
00133 // Inserindo músicas em uma segunda playlist...
00134 Playlist* p2 = new Playlist;
00135 p2->setName("Playlist 2");
00136
00137 tempSong.setTitle("Running in the 90s");
00138 tempSong.setArtist("Max Coveri");
00139 p2->insertSong(1, tempSong);
00140
00141 tempSong.setTitle("It's My Life");
00142 tempSong.setArtist("No Doubt");
00143 p2->insertSong(2, tempSong);
00144
00145 cout << p2->getName() << ":" << endl;
00146 p2->displayAllSongs(p2->getSongs()->getHead());
00147 cout << endl;
00148 // Inserindo músicas de p1 em p2 usando sobrecarga...
00149 p2->insertSong(*p1);
00150 cout << p2->getName() << " com as músicas da playlist 1:" << endl;
00151 p2->displayAllSongs(p2->getSongs()->getHead());
00152 cout << endl << "_____ " << endl << endl;
00153
00154 // Removendo de p2 as músicas de p1 usando sobrecarga...
00155 size_t count = p2->removeSong(*p1);
00156 cout << p2->getName() << " sem as " << count << " músicas da playlist 1:" << endl;
00157 p2->displayAllSongs(p2->getSongs()->getHead());
00158 cout << endl << "_____ " << endl << endl;
00159
00160 // Criando uma playlist p3 idêntica a p1 utilizando o construtor cópia...
00161 Playlist* p3 = new Playlist(*p1);
00162 p3->setName("Playlist 3");
00163 cout << p3->getName() << " cópia de playlist 1:" << endl;
00164 p3->displayAllSongs(p3->getSongs()->getHead());
00165 cout << endl << "_____ " << endl << endl;
00166
00167 // Adicionando mais músicas a p1...
00168 tempSong.setTitle("Love Sux");
00169 tempSong.setArtist("Avril Lavigne");
00170 p1->insertSong(3, tempSong);
00171
00172 tempSong.setTitle("Why do I");
00173 tempSong.setArtist("Set it Off");
00174 p1->insertSong(4, tempSong);
00175
00176 cout << p1->getName() << " com mais músicas:" << endl;
00177 p1->displayAllSongs(p1->getSongs()->getHead());
00178 cout << endl << "_____ " << endl << endl;
00179
00180 // Criando uma playlist 4 resultante da união da playlist 1 e playlist 2...
00181 Playlist* p4 = new Playlist(*p1 + *p2);
00182 p4->setName("Playlist 4");
00183 cout << p4->getName() << " criada a partir da playlist 1 em união com a playlist 2:" << endl;
00184 p4->displayAllSongs(p4->getSongs()->getHead());
00185 cout << endl << "_____ " << endl << endl;
00186
00187 // Criando uma playlist p5 usando a playlist 4 e o "tempSong"...

```

```

00188     tempSong.setTitle("Fooled Again");
00189     tempSong.setArtist("Lonely Bunker");
00190     Playlist* p5 = new Playlist(*p4 + tempSong);
00191     p5->setName("Playlist 5");
00192     cout << p5->getName() << " criada a partir da playlist 4 + uma nova música:" << endl;
00193     p5->displayAllSongs(p5->getSongs()->getHead());
00194     cout << endl << " _____ " << endl << endl;
00195
00196     // Criando a playlist 6 baseando se na playlist 5 menos as músicas da playlist 3...
00197     Playlist* p6 = new Playlist(*p5 - *p3);
00198     p6->setName("Playlist 6");
00199     cout << p6->getName() << " criada da playlist 5 - playlist 3:" << endl;
00200     p6->displayAllSongs(p6->getSongs()->getHead());
00201     cout << endl << " _____ " << endl << endl;
00202
00203     // Criando a playlist 7 a partir da playlist 6 menos o "tempSong"...
00204     Playlist* p7 = new Playlist(*p6 - tempSong);
00205     p7->setName("Playlist 7");
00206     cout << p7->getName() << " criada da playlist 6 - uma música:" << endl;
00207     p7->displayAllSongs(p7->getSongs()->getHead());
00208     cout << endl << " _____ " << endl << endl;
00209
00210     // Extraíndo a última música da playlist 6...
00211     Song* songPtr = new Song;
00212     *p6 » songPtr;
00213
00214     cout << p6->getName() << " após extração:" << endl;
00215     p6->displayAllSongs(p6->getSongs()->getHead());
00216     cout << endl << "Música que foi extraída: ";
00217     cout << songPtr->getTitle() << " - " << songPtr->getArtist() << endl;
00218     cout << endl << " _____ " << endl << endl;
00219
00220     // Adicionando novamente a música ao fim da playlist 6...
00221     *p6 << songPtr;
00222     cout << p6->getName() << " após inserção:" << endl;
00223     p6->displayAllSongs(p6->getSongs()->getHead());
00224
00225     cout << endl << " _____ " << endl << endl;
00226     cout << "Eliminando variáveis..." << endl;
00227     //Deletando as listas e variáveis do teste...
00228     cout << "testList: ";
00229     delete testList;
00230     cout << "Concluído!" << endl;
00231     cout << "Lista 1: ";
00232     delete lista1;
00233     cout << "Concluído!" << endl;
00234     cout << "Lista 2: ";
00235     delete lista2;
00236     cout << "Concluído!" << endl;
00237     cout << "Lista 3: ";
00238     delete lista3;
00239     cout << "Concluído!" << endl;
00240     cout << "Playlist 1: ";
00241     delete p1;
00242     cout << "Concluído!" << endl;
00243     cout << "Playlist 2: ";
00244     delete p2;
00245     cout << "Concluído!" << endl;
00246     cout << "Playlist 3: ";
00247     delete p3;
00248     cout << "Concluído!" << endl;
00249     cout << "Playlist 4: ";
00250     delete p4;
00251     cout << "Concluído!" << endl;
00252     cout << "Playlist 5: ";
00253     delete p5;
00254     cout << "Concluído!" << endl;
00255     cout << "Playlist 6: ";
00256     delete p6;
00257     cout << "Concluído!" << endl;
00258     cout << "Playlist 7: ";
00259     delete p7;
00260     cout << "Concluído!" << endl;
00261     cout << "temp: ";
00262     delete temp;
00263     cout << "Concluído!" << endl;
00264     cout << "songPtr: ";
00265     delete songPtr;
00266     cout << "Concluído!" << endl << endl;
00267     cout << "Teste finalizado com sucesso!" << endl;
00268 }

```


5.22.2.6 toLowercase()

```
string toLowercase (
    string s )
```

Essa função recebe uma string, e percorre-a transformando todos os caracteres em minúsculos.

Parâmetros

s	é a string a ser transformada.
---	--------------------------------

Retorna

o resultado da conversão.

Definição na linha 23 do arquivo `utilitarios.cpp`.

```
00023 {
00024     string result = "";
00025     for (size_t i = 0; i < s.size(); i++) {
00026         result += tolower(s[i]);
00027     }
00028     return result;
00029 }
```

5.23 utilitarios.cpp

Vá para a documentação desse arquivo.

```
00001
00008 #include <iostream>
00009 #include <cctype>
00010 #include "utilitarios.h"
00011 #include "musica.h"
00012 #include "listaLigada.h"
00013 #include "playlist.h"
00014 #include "listaPlaylists.h"
00015
00016 using namespace std;
00017
00023 string toLowercase(string s) {
00024     string result = "";
00025     for (size_t i = 0; i < s.size(); i++) {
00026         result += tolower(s[i]);
00027     }
00028     return result;
00029 }
00030
00036 string checkInt(string s){
00037     string aux = "-1";
00038     if (isdigit(s[0]) == true) {
00039         return s;
00040     } else {
00041         return aux;
00042     }
00043 }
00044
00048 void testeSobrecarga(){
00049     LinkedList* testList = new LinkedList;
00050     Song tempSong;
00051     LinkedList* listal = new LinkedList; //Criando uma lista ligada...
00052     // Inserindo músicas previamente a uma lista...
00053
00054     // Adicionando primeira música
00055     tempSong.setTitle("Shelter");
00056     tempSong.setArtist("Porter Robinson");
00057     listal->insertEnd(tempSong);
00058     // Adicionando segunda música
00059     tempSong.setTitle("Never Gonna Give You Up");
00060     tempSong.setArtist("Rick Astley");
00061     listal->insertEnd(tempSong);
00062     //Imprimindo a lista...
```

```

00063     cout << "Lista 1:" << endl;
00064     lista1->display();
00065     cout << endl << " _____ " << endl << endl;
00066
00067     // Cria uma segunda lista utilizando o construtor cópia.
00068     LinkedList* lista2 = new LinkedList(*lista1);
00069
00070     // Adiciona mais músicas a segunda lista
00071     tempSong.setTitle("Live and Learn");
00072     tempSong.setArtist("Crush 40");
00073     lista2->insertEnd(tempSong);
00074
00075     tempSong.setTitle("Bone Dry");
00076     tempSong.setArtist("Tristan");
00077     lista2->insertEnd(tempSong);
00078
00079     tempSong.setTitle("Lua de Cristal");
00080     tempSong.setArtist("Xuxa Meneghel");
00081     lista2->insertEnd(tempSong);
00082
00083     cout << "Lista 2 criada a partir da lista 1 com mais músicas:" << endl;
00084     lista2->display();
00085     cout << endl << " _____ " << endl << endl;
00086
00087     lista2->removePosition(*lista1); //Removendo da lista 2 as músicas da lista 1 por meio de
sobrecarga...
00088
00089     cout << "Lista 2 sem os elementos da lista 1:" << endl;
00090     lista2->display();
00091
00092     //Inserindo as músicas dessa lista à lista global por meio de sobrecarga...
00093     testList->insertEnd(*lista2);
00094     cout << endl << " _____ " << endl << endl;
00095
00096     // Criando uma nova lista a partir da concatenação das duas listas já existentes...
00097     LinkedList* lista3 = new LinkedList(*lista1 + *lista2);
00098     cout << "Lista 3 criada usando a lista 1 + lista 2: " << endl;
00099     lista3->display();
00100     cout << endl << " _____ " << endl << endl;
00101
00102     // Extraindo o ultimo nó da lista 3...
00103     no* temp = new no;
00104     *lista3 » temp;
00105
00106     cout << "Lista 3 pós-extração:" << endl;
00107     lista3->display();
00108     cout << endl << "Música extraída: ";
00109     cout << temp->data.getTitle() << " - " << temp->data.getArtist() << endl;
00110     cout << endl << " _____ " << endl << endl;
00111
00112     // Retornando o nó final da lista 3...
00113     *lista3 << temp;
00114     cout << "Lista 3 após inserção:" << endl;
00115     lista3->display();
00116     cout << endl << " _____ " << endl << endl;
00117
00118     // Inserindo músicas previamente a uma playlist
00119     Playlist* p1 = new Playlist;
00120     p1->setName("Playlist 1");
00121
00122     tempSong.setTitle("Never Gonna Give You Up");
00123     tempSong.setArtist("Rick Astley");
00124     p1->insertSong(1, tempSong);
00125
00126     tempSong.setTitle("Shelter");
00127     tempSong.setArtist("Porter Robinson");
00128     p1->insertSong(2, tempSong);
00129
00130     cout << p1->getName() << ":" << endl;
00131     p1->displayAllSongs(p1->getSongs()->getHead());
00132     cout << endl << " _____ " << endl << endl;
00133     // Inserindo músicas em uma segunda playlist...
00134     Playlist* p2 = new Playlist;
00135     p2->setName("Playlist 2");
00136
00137     tempSong.setTitle("Running in the 90s");
00138     tempSong.setArtist("Max Coveri");
00139     p2->insertSong(1, tempSong);
00140
00141     tempSong.setTitle("It's My Life");
00142     tempSong.setArtist("No Doubt");
00143     p2->insertSong(2, tempSong);
00144
00145     cout << p2->getName() << ":" << endl;
00146     p2->displayAllSongs(p2->getSongs()->getHead());
00147     cout << endl;
00148     // Inserindo músicas de p1 em p2 usando sobrecarga...

```

```

00149     p2->insertSong(*p1);
00150     cout << p2->getName() << " com as músicas da playlist 1:" << endl;
00151     p2->displayAllSongs(p2->getSongs()->getHead());
00152     cout << endl << " _____" << endl << endl;
00153
00154     // Removendo de p2 as músicas de p1 usando sobrecarga...
00155     size_t count = p2->removeSong(*p1);
00156     cout << p2->getName() << " sem as " << count << " músicas da playlist 1:" << endl;
00157     p2->displayAllSongs(p2->getSongs()->getHead());
00158     cout << endl << " _____" << endl << endl;
00159
00160     // Criando uma playlist p3 idêntica a p1 utilizando o construtor cópia...
00161     Playlist* p3 = new Playlist(*p1);
00162     p3->setName("Playlist 3");
00163     cout << p3->getName() << " cópia de playlist 1:" << endl;
00164     p3->displayAllSongs(p3->getSongs()->getHead());
00165     cout << endl << " _____" << endl << endl;
00166
00167     // Adicionando mais músicas a p1...
00168     tempSong.setTitle("Love Sux");
00169     tempSong.setArtist("Avril Lavigne");
00170     p1->insertSong(3, tempSong);
00171
00172     tempSong.setTitle("Why do I");
00173     tempSong.setArtist("Set it Off");
00174     p1->insertSong(4, tempSong);
00175
00176     cout << p1->getName() << " com mais músicas:" << endl;
00177     p1->displayAllSongs(p1->getSongs()->getHead());
00178     cout << endl << " _____" << endl << endl;
00179
00180     // Criando uma playlist 4 resultante da união da playlist 1 e playlist 2...
00181     Playlist* p4 = new Playlist(*p1 + *p2);
00182     p4->setName("Playlist 4");
00183     cout << p4->getName() << " criada a partir da playlist 1 em união com a playlist 2:" << endl;
00184     p4->displayAllSongs(p4->getSongs()->getHead());
00185     cout << endl << " _____" << endl << endl;
00186
00187     // Criando uma playlist p5 usando a playlist 4 e o "tempSong"...
00188     tempSong.setTitle("Fooled Again");
00189     tempSong.setArtist("Lonely Bunker");
00190     Playlist* p5 = new Playlist(*p4 + tempSong);
00191     p5->setName("Playlist 5");
00192     cout << p5->getName() << " criada a partir da playlist 4 + uma nova música:" << endl;
00193     p5->displayAllSongs(p5->getSongs()->getHead());
00194     cout << endl << " _____" << endl << endl;
00195
00196     // Criando a playlist 6 baseando se na playlist 5 menos as músicas da playlist 3...
00197     Playlist* p6 = new Playlist(*p5 - *p3);
00198     p6->setName("Playlist 6");
00199     cout << p6->getName() << " criada da playlist 5 - playlist 3:" << endl;
00200     p6->displayAllSongs(p6->getSongs()->getHead());
00201     cout << endl << " _____" << endl << endl;
00202
00203     // Criando a playlist 7 a partir da playlist 6 menos o "tempSong"...
00204     Playlist* p7 = new Playlist(*p6 - tempSong);
00205     p7->setName("Playlist 7");
00206     cout << p7->getName() << " criada da playlist 6 - uma música:" << endl;
00207     p7->displayAllSongs(p7->getSongs()->getHead());
00208     cout << endl << " _____" << endl << endl;
00209
00210     // Extraíndo a última música da playlist 6...
00211     Song* songPtr = new Song;
00212     *p6 >> songPtr;
00213
00214     cout << p6->getName() << " após extração:" << endl;
00215     p6->displayAllSongs(p6->getSongs()->getHead());
00216     cout << endl << "Música que foi extraída: ";
00217     cout << songPtr->getTitle() << " - " << songPtr->getArtist() << endl;
00218     cout << endl << " _____" << endl << endl;
00219
00220     // Adicionando novamente a música ao fim da playlist 6...
00221     *p6 << songPtr;
00222     cout << p6->getName() << " após inserção:" << endl;
00223     p6->displayAllSongs(p6->getSongs()->getHead());
00224
00225     cout << endl << " _____" << endl << endl;
00226     cout << "Eliminando variáveis..." << endl;
00227     //Deletando as listas e variáveis do teste...
00228     cout << "testList: ";
00229     delete testList;
00230     cout << "Concluído!" << endl;
00231     cout << "Lista 1: ";
00232     delete lista1;
00233     cout << "Concluído!" << endl;
00234     cout << "Lista 2: ";
00235     delete lista2;

```

```

00236     cout << "Concluído!" << endl;
00237     cout << "Lista 3: ";
00238     delete lista3;
00239     cout << "Concluído!" << endl;
00240     cout << "Playlist 1: ";
00241     delete p1;
00242     cout << "Concluído!" << endl;
00243     cout << "Playlist 2: ";
00244     delete p2;
00245     cout << "Concluído!" << endl;
00246     cout << "Playlist 3: ";
00247     delete p3;
00248     cout << "Concluído!" << endl;
00249     cout << "Playlist 4: ";
00250     delete p4;
00251     cout << "Concluído!" << endl;
00252     cout << "Playlist 5: ";
00253     delete p5;
00254     cout << "Concluído!" << endl;
00255     cout << "Playlist 6: ";
00256     delete p6;
00257     cout << "Concluído!" << endl;
00258     cout << "Playlist 7: ";
00259     delete p7;
00260     cout << "Concluído!" << endl;
00261     cout << "temp: ";
00262     delete temp;
00263     cout << "Concluído!" << endl;
00264     cout << "songPtr: ";
00265     delete songPtr;
00266     cout << "Concluído!" << endl << endl;
00267     cout << "Teste finalizado com sucesso!" << endl;
00268 }
00269
00273 void helpPage() {
00274
00275     cout << "Comandos para gerenciamento de músicas: " << endl << endl;
00276
00277     cout << "add - Adicionar uma música" << endl;
00278     cout << "del - Remover uma música" << endl;
00279     cout << "list - Listar todas as músicas" << endl;
00280     cout << "search - Buscar uma música" << endl << endl;
00281
00282     cout << "Comandos para gerenciamento de playlists: " << endl << endl;
00283
00284     cout << "addp - Adicionar uma playlist" << endl;
00285     cout << "delp - Remover uma playlist" << endl;
00286     cout << "listp - Listar todas as playlists" << endl << endl;
00287
00288     cout << "Comandos para reproduzir músicas de playlists: " << endl;
00289
00290     cout << "playp - Comece a tocar uma playlist" << endl;
00291     cout << "playn - Toque a próxima música de uma playlist." << endl;
00292     cout << "playb - Volte uma música." << endl;
00293     cout << "plays - Pare a reprodução de músicas." << endl << endl;
00294
00295     cout << "Comandos para gerenciamento de músicas em playlists: " << endl << endl;
00296
00297     cout << "addmp - Adicionar música a uma playlist" << endl;
00298     cout << "delmp - Remover música de uma playlist" << endl;
00299     cout << "mmp - Mover música numa playlist" << endl;
00300     cout << "listmp - Listar músicas de uma playlist" << endl;
00301     cout << "copyp - Criar uma cópia de uma playlist existente." << endl;
00302     cout << "mergep - Unir duas playlists em uma única playlist" << endl;
00303     cout << "mergep- - Remover músicas da segunda playlist da primeira e retornar uma nova playlist" <<
endl << endl;
00304     cout << "savetf - Salvar uma playlist em um arquivo" << endl;
00305
00306     cout << "Comandos para teste:" << endl;
00307
00308     cout << "addtp - Adicionar músicas e playlists pré-definidas ao programa." << endl;
00309     cout << "otest - Teste de métodos sobrecarregados." << endl << endl;
00310
00311     cout << "quit - Encerrar o programa." << endl;
00312     cout << "_____ " << endl << endl;
00313 }
00314
00318 void telaInicial() {
00319     cout << "        ----- Tela Inicial -----" << endl;
00320     cout << "Para uma lista de comandos digite 'help' " << endl << endl;
00321     cout << "Digite um comando: ";
00322 }
00323
00330 void stopPlaying(int& playing, size_t& index){
00331     if(playing == 0){
00332         cout << "Nada está tocando no momento..." << endl;
00333     }

```

```
00334     if(playing == 1){
00335         cout << "Parando a reprodução!" << endl;
00336         index = 0;
00337         playing = 0;
00338     }
00339 }
```

