

## Musicas e Playlists

v0.1

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 list 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.
# playp - Comando para reproduzir uma playlist.
```

### 1.0.5 Comandos para gerenciamento de música nas playlists:

```
# 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.
# 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.
```

### 1.0.6 Exemplo de utilização do programa:

#### 1.0.6.1 Iniciando o programa

```
$ ./music
```

#### 1.0.6.2 Tela inicial do programa:

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

Digite um comando:
```

#### 1.0.6.3 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
playp - Comece a tocar uma playlist

Comandos para gerenciamento de músicas em playlists:

playn - Toque a próxima música de uma playlist.
playb - Volte uma música.
plays - Pare a reprodução de músicas.
addmp - Adicionar música a uma playlist
delmp - Remover música de uma playlist
mmp - Mover música numa playlist
listmp - Listar músicas de uma playlist

Comando para encerrar o programa:

quit - Sair

-----
```

#### 1.0.6.4 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!

### 1.0.6.5 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.

### 1.0.6.6 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

### 1.0.6.7 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!

### 1.0.6.8 Adicionando uma playlist

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

Digite um comando: addp

-----  
Nome da playlist: Favoritas

Playlist criada com sucesso!

### 1.0.6.9 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.
```

### 1.0.6.10 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
```

### 1.0.6.11 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
```

### 1.0.6.12 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
```

### 1.0.6.13 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
```

#### 1.0.6.14 Parar a reprodução de músicas

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

Digite um comando: plays

-----

Parando a reprodução!
```

#### 1.0.6.15 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'
```

#### 1.0.6.16 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'
```

#### 1.0.6.17 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.

### 1.0.6.18 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
```

### 1.0.6.19 Encerrando o programa

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

Digite um comando: quit
```

```
-----

Encerrando o programa.
```

A documentação também pode ser encontrada em html na pasta `"/docs/html/index.html"` que está incluída no arquivo ZIP deste programa.



## Chapter 2

# Índice das Estruturas de Dados

### 2.1 Estruturas de Dados

Aqui estão as estruturas de dados, uniões e suas respectivas descrições:

<a href="#">LinkedList</a>	??
<a href="#">ListOfPlaylists</a>	??
<a href="#">no</a>	??
<a href="#">no_</a>	??
<a href="#">Playlist</a>	??
<a href="#">Song</a>	??



## Chapter 3

# Índice dos Arquivos

### 3.1 Lista de Arquivos

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

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



## Chapter 4

# Estruturas

### 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 \(\)](#)

#### 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 ( )
```

Definição na linha 146 do arquivo [listaPlaylists.cpp](#).

```
00146 {  
00147     no_* temp = head;  
00148     size_t i = 1;  
00149     string nome = "";  
00150     while (temp != nullptr) {  
00151         nome = temp->data->getName();  
00152         cout << i << " - " << nome << endl;  
00153         temp = temp->next;  
00154         ++i;  
00155     }  
00156 }
```

### 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 <a href="#">Song</a> com a música a ser removida. Essa função mostra todas as playlists armazenadas na lista ligada.
---------------	--

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

#### 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 Campos

### 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:

#### Campos de Dados

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

#### 4.3.1 Descrição detalhada

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

#### 4.3.2 Campos

##### 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\_:

### Campos de Dados

- [Playlist \\* data](#)
- [no\\_ \\* next](#)

#### 4.4.1 Descrição detalhada

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

#### 4.4.2 Campos

##### 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 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 13 do arquivo [playlist.cpp](#).

#### 4.5.2.2 ~Playlist()

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

Retorna as músicas de uma playlist

Definição na linha 19 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 42 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 201 do arquivo [playlist.cpp](#).

```
00201                                     {
00203     if (current == nullptr) {
00204         count = 1;
00205         return;
00206     }
00207     cout << count << " - " << current->data.getTitle() << " - " << current->data.getArtist() << endl;
00208     ++count;
00209     displayAllSongs(current->next);
00210 }
```

#### 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 217 do arquivo [playlist.cpp](#).

```
00217                                     {
00218     if (current == nullptr) {
00219         count = 1;
00220         return;
00221     }
00222     if(count == pos){
00223         cout << current->data.getTitle() << " - " << current->data.getArtist() << endl;
00224     }
00225     ++count;
00226     displayOne(current->next, pos);
00227 }
```

#### 4.5.3.3 getName()

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

Define o nome de uma playlist

Definição na linha 27 do arquivo [playlist.cpp](#).

#### 4.5.3.4 getSongs()

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

Retorna o nome de uma playlist

Definição na linha 23 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 163 do arquivo `playlist.cpp`.

```
00163
00165     if (toInsert.getSongs()->getSize() < 1) {
00166         return;
00167     } else {
00168         songs->insertEnd(*toInsert.getSongs());
00169     }
00170 }
```

## 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 <code>Song</code> .

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

Definição na linha 124 do arquivo `playlist.cpp`.

```
00124
00125     songs->insertPosition(pos, value);
00126     playing = songs->getHead();
00127 }
```

## 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 142 do arquivo [playlist.cpp](#).

```
00142                                     {
00143     if (start != end) {
00144         no* target = songs->getno(start);
00145         Song value = target->data;
00146         songs->removePosition(start);
00147         if (end < start) {
00148             songs->insertPosition(end, value);
00149         } else {
00150             songs->insertPosition(end + 1, value);
00151         }
00152         playing = songs->getHead();
00153     }
00154 }
```

#### 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 35 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 54 do arquivo [playlist.cpp](#).

```
00054                                     {
00055     Playlist resultante;
00056     resultante.insertSong(*this);
00057     resultante.songs->insertEnd(toAdd);
00058     return resultante;
00059 }
```



#### 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 <a href="#">Playlist</a> .
--------------------	---

##### 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 66 do arquivo [playlist.cpp](#).

```
00066 {
00067     Playlist resultante(*this);
00068     no* temp = segPlaylist.getSongs()->getHead();
00070     while (temp != nullptr) {
00072         size_t pos = resultante.getSongs()->getPosition(temp->data);
00073         if (pos > 0) {
00074             resultante.removeSong(pos);
00075         }
00076         temp = temp->next;
00077     }
00078     return resultante;
00079 }
```

#### 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 <a href="#">Song</a> .
-----------------	---

##### 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 85 do arquivo `playlist.cpp`.

```
00085 {
00086     Playlist resultante(*this);
00087     size_t pos = resultante.getSongs()->getPosition(toRemove);
00089     if (pos > 0) {
00090         resultante.removeSong(pos);
00091     }
00092     return resultante;
00093 }
```

#### 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 112 do arquivo `playlist.cpp`.

```
00112 {
00113     if (newSong == nullptr) {
00114         return;
00115     } else {
00116         songs->insertEnd(*newSong);
00117     }
00118 }
```

#### 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 98 do arquivo `playlist.cpp`.

```

00098                                     {
00099     if (songs->getSize() > 0) {
00100         no* temp = new no;
00101         *songs » temp;
00102         *lastSong = temp->data;
00103         delete temp;
00104     } else {
00105         lastSong = nullptr;
00106     }
00107 }

```

#### 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 177 do arquivo [playlist.cpp](#).

```

00177                                     {
00179     if (toRemove.getSongs()->getSize() < 1) {
00180         return 0;
00181     } else {
00182         size_t removed = 0;
00183         no* temp = toRemove.getSongs()->getHead();
00184         while (temp != nullptr) {
00185             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 }

```

#### 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 133 do arquivo [playlist.cpp](#).

```
00133 {
00134     songs->removePosition(pos);
00135 }
```

#### 4.5.3.17 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 31 do arquivo [playlist.cpp](#).

### 4.5.4 Campos

#### 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 Campos

### 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:

#### Estruturas de Dados

- 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 início 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:

#### Estruturas de Dados

- 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:

### Estruturas de Dados

- 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:

### Estruturas de Dados

- 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 displayOne(no* current, int pos);
00046 };
00047
00048 #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)

### 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 31 do arquivo [utilitarios.cpp](#).

```
00031 {
00032     string aux = "-1";
00033     if (isdigit(s[0]) == true) {
00034         return s;
00035     } else {
00036         return aux;
00037     }
00038 }
```

### 5.9.1.2 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 18 do arquivo [utilitarios.cpp](#).

```
00018 {
00019     string result = "";
00020     for (size_t i = 0; i < s.size(); i++) {
00021         result += tolower(s[i]);
00022     }
00023     return result;
00024 }
```

## 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 #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()) ) {
00369             return pos;
00370         }
00371         temp = temp->next;
00372         ++pos;
00373     }
00374     return 0;
00375 }
00376
00377
00379 void LinkedList::display() {
00380     no* temp = head;
00381     size_t index = 1;
00382
00383     while (temp != nullptr) {
00384         Song s = temp->data;
00385         cout << index << " - " << s.getTitle() << " - " << s.getArtist() << endl;
00386         temp = temp->next;
00387         ++index;
00388     }
00389 }

```

## 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 }
00054
00055 Playlist* ListOfPlaylists::searchPlaylist(string searchName) {
00056     no_* temp = head;
00057     while (temp != nullptr) {
00058         if ( toLowercase(temp->data->getName()) == toLowercase(searchName) ) {
00059             return temp->data;
00060         }
00061         temp = temp->next;
00062     }
00063     return nullptr;
00064 }
00065
00067 void ListOfPlaylists::insertPlaylist(Playlist* value) {
00068     if (searchPlaylist(value->getName()) != nullptr) {
00069         cout << "Uma playlist com esse nome já existe!" << endl << endl;
00070     } else {
00071         no_* temp = new no_;
00072         temp->data = value;
00073         temp->next = nullptr;
00074         if (head == nullptr) {
00075             head = temp;
00076             tail = temp;
00077             temp = nullptr;
00078         } else {
00079             tail->next = temp;
00080             tail = temp;
00081         }
00082         ++size;
00083         cout << endl << "Playlist criada com sucesso!" << endl;
00084     }
00085 }
00086
00088 void ListOfPlaylists::removePlaylist(size_t pos) {
00089     no_* pre = nullptr;
00090     no_* cur = nullptr;
00091
00092     cur = head;
00093     if (pos == 1) {
00094         head = head->next;
00095     } else if (pos < size) {
00096         for (size_t i = 1; i < pos; ++i) {
00097             pre = cur;
00098             cur = cur->next;
00099         }
00100         pre->next = cur->next;
00101     }
00102     delete cur;
00103     --size;
00104 }

```

```

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 }
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 }
00146 void ListOfPlaylists::display() {
00147     no_* temp = head;
00148     size_t i = 1;
00149     string nome = "";
00150     while (temp != nullptr) {
00151         nome = temp->data->getName();
00152         cout << i << " - " << nome << endl;
00153         temp = temp->next;
00154         ++i;
00155     }
00156 }

```

## 5.16 Referência do Arquivo src/main.cpp

Projeto que organiza músicas e playlists em listas ligadas.

```

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

```

Gráfico de dependência de inclusões para main.cpp:

### Funções

- void [telainicial](#) ()
- void [helpPage](#) ()
- int [main](#) (int argc, char const \*argv[])

### 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 helpPage()

```
void helpPage ( )
```

Imprime a tela inicial.

Definição na linha [533](#) do arquivo [main.cpp](#).

#### 5.16.2.2 main()

```
int main (
    int argc,
    char const * argv[] )
```

Imprime os comandos que o programa pode executar.

Definição na linha [26](#) do arquivo [main.cpp](#).

#### 5.16.2.3 telaInicial()

```
void telaInicial ( )
```

Definição na linha [561](#) do arquivo [main.cpp](#).

```
00561     {
00562         cout << "          ----- Tela Inicial -----" << endl;
00563         cout << "Para uma lista de comandos digite 'help' " << endl << endl;
00564         cout << "Digite um comando: ";
00565     }
```

### 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
00012 #include "musica.h"
00013 #include "listaLigada.h"
00014 #include "utilitarios.h"
00015 #include "playlist.h"
00016 #include "listaPlaylists.h"
00017
00018 using namespace std;
00019
00020 void telaInicial();
00021 void helpPage();
00022
00023 string chkint;
00024 int playing = 0;
00025
00026 int main(int argc, char const *argv[])
00027 {
00028     // Cria a lista global de músicas do programa
00029     LinkedList* globalList = new LinkedList;
00030     // Cria a lista de armazenar as playlists do programa
00031     ListOfPlaylists* playlists = new ListOfPlaylists;
00032
00033     size_t option = 12; // Opção selecionada no menu
00034     no* searchResult = nullptr; // Ponteiro para o nó retornado na busca
00035     size_t index = 0; // Índice na lista para as interações com o usuário
00036     size_t index2 = 0; // Índice na lista para as interações com o usuário
00037     // Variáveis temporárias para armazenar as entradas do usuário
00038     Playlist* tempPlaylist = nullptr;
00039     Song tempSong;
00040     string tempTitle = "";
00041     string tempArtist = "";
00042     // Executa o menu de funcionalidades enquanto a opção for diferente de 0
00043     while (option != 0) {
00044         // Limpa a tela e exibe as opções do menu
00045         int validcommand = 0;
00046         telaInicial();
00047         string chooser;
00048         //Le a opção escolhida
00049         cin >> chooser;
00050         cout << endl << "-----" << endl << endl;
00051         if(chooser == "add") { // Adicionar uma música
00052             validcommand = 1;
```

```

00053         // Le as entradas
00054         cout << "Título da música: ";
00055         cin.ignore(256, '\n'); //Ignorar o "Enter"
00056         getline(cin, tempTitle);
00057         cout << "Nome do artista: ";
00058         getline(cin, tempArtist);
00059
00060         // Atribui num objeto do tipo Song
00061         tempSong.setTitle(tempTitle);
00062         tempSong.setArtist(tempArtist);
00063
00064         // Adiciona ao final da lista global
00065         int endadded = globalList->insertEnd(tempSong);
00066         if (endadded == 1){
00067             cout << endl << "Música adicionada com sucesso!" << endl << endl;
00068         }
00069     }
00070     if(chooser == "del"){ // Remover uma música
00071         validcommand = 1;
00072         if (globalList->getSize() == 0) {
00073             cout << "Não existem músicas registradas..." << endl << endl;
00074         } else {
00075             // Imprime a lista de músicas com seus índices
00076             globalList->display();
00077
00078             // Lê a entrada
00079             cout << endl << "Insira o índice da música a ser removida (enumerada acima): ";
00080             cin >> chkint; //O input do usuário irá para chkint uma variável do tipo string.
00081             index = stoi(checkInt(chkint)); //Caso o valor em chkint seja de fato um inteiro, index
00082             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()
00092             << " foi deletada." << endl;
00093
00094             // Remove de todas as playlists
00095             if (playlists->getSize() > 0) {
00096                 playlists->removeFromAll(searchResult->data);
00097             }
00098
00099             // Remove da lista global
00100             globalList->removePosition(index);
00101         }
00102     }
00103     if(chooser == "list"){ // Listar todas as músicas
00104         validcommand = 1;
00105         // Processo para garantir que há músicas adicionadas
00106         if (globalList->getSize() == 0) {
00107             cout << "Ainda não há nada aqui. Experimente adicionar algumas músicas." << endl << endl;
00108         } else {
00109             cout << "Músicas armazenadas:" << endl << endl;
00110             // Imprime a lista de músicas com seus índices
00111             globalList->display();
00112             cout << endl;
00113         }
00114     }
00115     if(chooser == "playp"){
00116         // Processo para garantir que há playlists adicionadas
00117         validcommand = 1;
00118         if (playlists->getSize() == 0) {
00119             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma!" << endl;
00120         } else {
00121             // Lê a entrada
00122             cout << "Playlists disponíveis: " << endl;
00123             playlists->display();
00124             cout << "Insira o índice da playlist desejada: ";
00125             cin >> chkint;
00126             index = stoi(checkInt(chkint));
00127             cout << endl << endl;
00128             // Processo para garantir que a entrada é válida
00129             while (index < 1 || index > playlists->getSize()) {
00130                 cout << "Índice inválido! Tente novamente: ";
00131                 cin >> chkint;
00132                 index = stoi(checkInt(chkint));
00133             }
00134
00135             // Obtém a playlist pelo índice
00136             tempPlaylist = playlists->getPlaylist(index);
00137
00138             // Processo para garantir que há músicas adicionadas
00139             if (tempPlaylist->getSongs()->getSize() == 0) {

```

```

00138         cout << "Ops, não há músicas nessa playlist, adicione algumas músicas e tente novamente! "
<< endl << endl;
00139     } else {
00140         // Imprime a lista de músicas com seus índices
00141         tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00142
00143         // Lê a entrada
00144         cout << "Insira o índice da música que será tocada (consulte a lista de músicas): ";
00145         cin >> chkint;
00146         index = stoi(checkInt(chkint));
00147
00148         // Processo para garantir que a entrada é válida
00149         while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00150             cout << "Índice inválido! Tente novamente: ";
00151             cin >> chkint;
00152             index = stoi(checkInt(chkint));
00153         }
00154         cout << endl << "Tocando agora: ";
00155         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00156         playing = 1;
00157     }
00158 }
00159 }
00160 if(chooser == "playn"){
00161     // Processo para garantir que há playlists adicionadas
00162     validcommand = 1;
00163     if (playing == 0){
00164         cout << "Nada está tocando no momento..." << endl << endl;
00165     }
00166     if (playing == 1){
00167         if (playlists->getSize() == 0) {
00168             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00169         } else {
00170             index++;
00171         }
00172         if (index > tempPlaylist->getSongs()->getSize()) {
00173             cout << endl << "Sua playlist acabou, mas a música nunca acaba, recomeçando! :D" << endl <<
endl;
00174             index = 1;
00175         }
00176         cout << "Tocando agora: ";
00177         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00178         cout << endl << endl;
00179     }
00180 }
00181 if(chooser == "playb"){
00182     // Processo para garantir que há playlists adicionadas
00183     validcommand = 1;
00184     if (playing == 0){
00185         cout << "Nada está tocando no momento..." << endl << endl;
00186     }
00187     if (playing == 1){
00188         if (playlists->getSize() == 0) {
00189             cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00190         } else {
00191             index--;
00192         }
00193         if (index < 1) {
00194             cout << endl << "Você está no início... Assim como o tempo, você não pode voltar atrás..." <<
endl << endl;
00195             index = 1;
00196         }
00197         cout << "Tocando agora: ";
00198         tempPlaylist->displayOne(tempPlaylist->getSongs()->getHead(), index);
00199         cout << endl << endl;
00200     }
00201 }
00202 if(chooser == "plays"){
00203     validcommand = 1;
00204     if(playing == 0){
00205         cout << "Nada está tocando no momento..." << endl;
00206     }
00207     if(playing == 1){
00208         cout << "Parando a reprodução!" << endl;
00209         index = 0;
00210         playing = 0;
00211     }
00212 }
00213 if(chooser == "search") { // Buscar uma música
00214     validcommand = 1;
00215     cout << "Título da música: ";
00216     cin.ignore(256, '\n');
00217     getline(cin, tempTitle);
00218     cout << "Nome do artista: ";
00219     getline(cin, tempArtist);
00220
00221     // Atribui num objeto do tipo Song

```



```

00222         tempSong.setTitle(tempTitle);
00223         tempSong.setArtist(tempArtist);
00224
00225         // Efetua a busca
00226         searchResult = globalList->search(tempSong);
00227
00228         // Da o retorno
00229         if (searchResult == nullptr) {
00230             cout << endl << "Essa música não foi adicionada! Verifique se você digitou corretamente ou
adicione a música antes de tentar novamente." << endl << endl;
00231         } else {
00232             cout << endl << "A música " << searchResult->data.getTitle() << " - " <<
searchResult->data.getArtist() << " está armazenada!" << endl;
00233         }
00234         cout << endl;
00235     }
00236     if(chooser == "addp") { // Adicionar uma playlist
00237         validcommand = 1;
00238         cout << "Nome da playlist: ";
00239         cin.ignore(256, '\n');
00240         getline(cin, tempTitle);
00241
00242         // Aloca a nova playlist
00243         tempPlaylist = new Playlist;
00244
00245         // Adiciona o nome escolhido
00246         tempPlaylist->setName(tempTitle);
00247
00248         // Insere na lista
00249         playlists->insertPlaylist(tempPlaylist);
00250
00251         cout << endl;
00252     }
00253     if(chooser == "delp") { // Remover uma playlist
00254         // Processo para garantir que há playlists adicionadas
00255         validcommand = 1;
00256         if (playlists->getSize() == 0) {
00257             cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl
<< endl;
00258         } else {
00259             // Imprime a lista de playlists com seus indices
00260             cout << "Playlists registradas: " << endl;
00261             playlists->display();
00262
00263             // Lê a entrada
00264             cout << endl << "Insira o índice da playlist a ser removida: ";
00265             cin >> chkint;
00266             index = stoi(checkInt(chkint));
00267             // Processo para garantir que a entrada é válida
00268             while (index < 1 || index > playlists->getSize()) {
00269                 cout << "Índice inválido! Tente novamente: ";
00270                 cin >> chkint;
00271                 index = stoi(checkInt(chkint));
00272             }
00273
00274             // Remove a playlist
00275             playlists->removePlaylist(index);
00276             cout << "Remoção concluída com sucesso." << endl << endl;
00277         }
00278     }
00279     if(chooser == "listp") { // Listar todas as playlists
00280         // Processo para garantir que as playlists foram adicionadas.
00281         validcommand = 1;
00282         if (playlists->getSize() == 0) {
00283             cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl
<< endl;
00284         } else {
00285             cout << "Playlists armazenadas atualmente:" << endl << endl;
00286             // Imprime a lista de músicas com seus indices
00287             playlists->display();
00288         }
00289     }
00290     if(chooser == "addmp"){ // Adicionar música a uma playlist
00291         // Processo para garantir que há playlists adicionadas
00292         validcommand = 1;
00293         if (playlists->getSize() == 0) {
00294             cout << endl << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl
<< endl;
00295         } else {
00296             // Lê a entrada
00297             cout << "Playlists disponíveis: " << endl;
00298             playlists->display();
00299             cout << "Insira o índice da playlist desejada (enumerada acima): ";
00300             int validstuff = 0;
00301             int invalidstuff = 0;
00302             while (validstuff == 0){
00303                 if(invalidstuff == 1){

```

```

00304         cout << "Índice inválido! Tente Novamente: ";
00305     }
00306     cin >> chkint;
00307     if (isdigit(chkint[0]) == true) {
00308         index = stoi(chkint);
00309         validstuff = 1;
00310     } else {
00311         validstuff = 0;
00312         invalidstuff = 1;
00313     }
00314 }
00315 // Processo para garantir que a entrada é válida
00316 while (index < 1 || index > playlists->getSize()) {
00317     cout << endl << "Índice inválido! Tente novamente: ";
00318     cin >> chkint;
00319     index = stoi(checkInt(chkint));
00320 }
00321
00322 // Obtém a playlist pelo índice
00323 tempPlaylist = playlists->getPlaylist(index);
00324 cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;
00325
00326 // Le as entradas
00327 cout << "Título da música: ";
00328 cin.ignore(256, '\n');
00329 getline(cin, tempTitle);
00330 cout << "Nome do artista: ";
00331 getline(cin, tempArtist);
00332 cout << "Em qual posição deseja adicionar (verifique a lista de músicas da playlist): ";
00333 validstuff = 0;
00334 while (validstuff == 0) {
00335     cin >> chkint;
00336     if (isdigit(chkint[0]) == true) {
00337         index = stoi(chkint);
00338         validstuff = 1;
00339     } else {
00340         cout << "Índice inválido! Tente novamente: ";
00341         validstuff = 0;
00342     }
00343 }
00344
00345 // Processo para garantir que a posição é válida
00346 while (index < 1) {
00347     cout << endl << "Posição inválida. Tente novamente: ";
00348     cin >> chkint;
00349     index = stoi(checkInt(chkint));
00350 }
00351 // Atribui num objeto do tipo Song
00352 tempSong.setTitle(tempTitle);
00353 tempSong.setArtist(tempArtist);
00354
00355 // Busca pela música armazenada
00356 searchResult = globalList->search(tempSong);
00357 if (searchResult == nullptr) {
00358     cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e
00359     tente novamente! " << endl << endl;
00360 } else {
00361     // Adiciona na playlist
00362     tempPlaylist->insertSong(index, tempSong);
00363     cout << endl << "Música adicionada à playlist '" << tempPlaylist->getName() << "' " << endl <<
00364     endl;
00365 }
00366 }
00367 if(chooser == "delmp"){ // Remover música de uma playlist
00368     // Processo para garantir que há playlists adicionadas
00369     validcommand = 1;
00370     if (playlists->getSize() == 0) {
00371         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma!" << endl << endl;
00372     } else {
00373         // Lê a entrada
00374         cout << "Playlists disponíveis: " << endl;
00375         playlists->display();
00376         cout << "Insira o índice da playlist desejada: ";
00377         cin >> chkint;
00378         index = stoi(checkInt(chkint));
00379
00380         // Processo para garantir que a entrada é válida
00381         while (index < 1 || index > playlists->getSize()) {
00382             cout << endl << "Índice inválido! Tente novamente: ";
00383             cin >> chkint;
00384             index = stoi(checkInt(chkint));
00385         }
00386
00387         // Obtém a playlist pelo índice
00388         tempPlaylist = playlists->getPlaylist(index);
00389         cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;

```

```

00389
00390 // Processo para garantir que há músicas adicionadas
00391 if (tempPlaylist->getSongs()->getSize() == 0) {
00392     cout << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
novamente! " << endl << endl;
00393 } else {
00394     // Imprime a lista de músicas com seus índices
00395     tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00396
00397     // Lê a entrada
00398     cout << "Insira o índice da música a ser removida (consulte a lista de músicas): ";
00399     cin >> chkint;
00400     index = stoi(checkInt(chkint));
00401
00402     // Processo para garantir que a entrada é válida
00403     while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00404         cout << "Índice inválido! Tente novamente: ";
00405         cin >> chkint;
00406         index = stoi(checkInt(chkint));
00407     }
00408
00409     // Remove a música
00410     tempPlaylist->removeSong(index);
00411     cout << endl << "Música removida da playlist '" << tempPlaylist->getName() << "'" << endl <<
endl;
00412 }
00413 }
00414 }
00415 if(chooser == "mmp"){ // Mover música numa playlist
00416     // Processo para garantir que há playlists adicionadas
00417     validcommand = 1;
00418     if (playlists->getSize() == 0) {
00419         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma! " << endl;
00420     } else {
00421         // Lê a entrada
00422         cout << "Playlists disponíveis: " << endl;
00423         playlists->display();
00424         cout << "Insira o índice da playlist desejada: ";
00425         cin >> chkint;
00426         index = stoi(checkInt(chkint));
00427
00428         // Processo para garantir que a entrada é válida
00429         while (index < 1 || index > playlists->getSize()) {
00430             cout << endl << "Índice inválido! Tente novamente: ";
00431             cin >> chkint;
00432             index = stoi(checkInt(chkint));
00433         }
00434
00435         // Obtém a playlist pelo índice
00436         tempPlaylist = playlists->getPlaylist(index);
00437         cout << "Playlist selecionada: " << tempPlaylist->getName() << endl << endl;
00438
00439         // Processo para garantir que há músicas adicionadas
00440         if (tempPlaylist->getSongs()->getSize() == 0) {
00441             cout << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e tente
novamente! " << endl << endl;
00442         } else {
00443             // Imprime a lista de músicas com seus índices
00444             cout << endl << "Músicas nesta playlist: " << endl;
00445             tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00446
00447             // Lê a entrada
00448             cout << endl << "Insira o índice da música a ser movida: ";
00449             cin >> chkint;
00450             index = stoi(checkInt(chkint));
00451
00452             // Processo para garantir que a entrada é válida
00453             while (index < 1 || index > tempPlaylist->getSongs()->getSize()) {
00454                 cout << endl << "Índice inválido! Tente novamente: ";
00455                 cin >> chkint;
00456                 index = stoi(checkInt(chkint));
00457             }
00458
00459             // Lê a entrada
00460             cout << "Insira o índice da posição para qual deseja movê-la: ";
00461             cin >> chkint;
00462             index2 = stoi(checkInt(chkint));
00463
00464             // Processo para garantir que a entrada é válida
00465             while (index2 < 1 || index2 > tempPlaylist->getSongs()->getSize()) {
00466                 cout << endl << "Índice inválido! Tente novamente: ";
00467                 cin >> chkint;
00468                 index2 = stoi(checkInt(chkint));
00469             }
00470
00471             // Move a música
00472             tempPlaylist->moveSong(index, index2);

```

```

00473         cout << endl << "Posição alterada com sucesso." << endl << endl;
00474     }
00475 }
00476 }
00477 }
00478 }
00479 if(chooser == "listmp") { // Listar músicas de uma playlist
00480     // Processo para garantir que há playlists adicionadas
00481     validcommand = 1;
00482     if (playlists->getSize() == 0) {
00483         cout << "Ops, nenhuma playlist foi adicionada até o momento, tente criar uma!" << endl;
00484     } else {
00485         // Lê a entrada
00486         cout << "Playlists disponíveis: " << endl;
00487         playlists->display();
00488         cout << "Insira o índice da playlist desejada: ";
00489         cin >> chkint;
00490         index = stoi(checkInt(chkint));
00491
00492         // Processo para garantir que a entrada é válida
00493         while (index < 1 || index > playlists->getSize()) {
00494             cout << "Índice inválido! Tente novamente: ";
00495             cin >> chkint;
00496             index = stoi(checkInt(chkint));
00497         }
00498
00499         // Obtém a playlist pelo índice
00500         tempPlaylist = playlists->getPlaylist(index);
00501
00502         // Processo para garantir que há músicas adicionadas
00503         if (tempPlaylist->getSongs()->getSize() == 0) {
00504             cout << endl << "Ops, a música não foi adicionada até o momento, por favor, adicione-a e
00505 tente novamente!" << endl << endl;
00506         } else {
00507             cout << endl << endl << "Músicas da playlist '" << tempPlaylist->getName() << "':" << endl <<
00508 endl;
00509             tempPlaylist->displayAllSongs(tempPlaylist->getSongs()->getHead());
00510         }
00511     }
00512 }
00513 if (chooser == "help"){
00514     validcommand = 1;
00515     helpPage();
00516 }
00517 if (chooser == "quit"){
00518     validcommand = 1;
00519     cout << "Encerrando o programa." << endl << endl;
00520     option = 0;
00521 }
00522 if(validcommand == 0){
00523     cout << "Comando inválido!" << endl << endl;
00524 }
00525 }
00526 // Libera a memória das listas globais
00527 delete globallist;
00528 delete playlists;
00529 return 0;
00530 }
00531 }
00532 void helpPage(){
00533     cout << "Comandos para gerenciamento de músicas: " << endl << endl;
00534     cout << "add - Adicionar uma música" << endl;
00535     cout << "del - Remover uma música" << endl;
00536     cout << "list - Listar todas as músicas" << endl;
00537     cout << "search - Buscar uma música" << endl << endl;
00538
00539     cout << "Comandos para gerenciamento de playlists: " << endl << endl;
00540     cout << "addp - Adicionar uma playlist" << endl;
00541     cout << "delp - Remover uma playlist" << endl;
00542     cout << "listp - Listar todas as playlists" << endl;
00543     cout << "playp - Comece a tocar uma playlist" << endl << endl;
00544
00545     cout << "Comandos para gerenciamento de músicas em playlists: " << endl << endl;
00546     cout << "playn - Toque a próxima música de uma playlist." << endl;
00547     cout << "playb - Volte uma música." << endl;
00548     cout << "plays - Pare a reprodução de músicas." << endl;
00549     cout << "addmp - Adicionar música a uma playlist" << endl;
00550     cout << "delmp - Remover música de uma playlist" << endl;
00551     cout << "mmp - Mover música numa playlist" << endl;
00552     cout << "listmp - Listar músicas de uma playlist" << endl << endl;
00553
00554     cout << "Comando para encerrar o programa: " << endl << endl;
00555     cout << "quit - Sair" << endl;
00556     cout << "-----" << endl << endl;
00557 }
00558 }

```

```
00559
00561 void telaInicial() {
00562     cout << "          ----- Tela Inicial -----" << endl;
00563     cout << "Para uma lista de comandos digite 'help' " << endl << endl;
00564     cout << "Digite um comando: ";
00565 }
```

## 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"
```

Gráfico de dependência de inclusões para playlist.cpp:

### 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](#).

## 5.21 playlist.cpp

Vá para a documentação desse arquivo.

```
00001
00008 #include "playlist.h"
00009
00010 using namespace std;
00011
00013 Playlist::Playlist() {
00014     songs = new LinkedList;
00015     playing = nullptr;
00016     count = 1;
00017 }
00019 Playlist::~~Playlist() {
00020     delete songs;
00021 }
00023 LinkedList* Playlist::getSongs() {
00024     return songs;
00025 }
00027 string Playlist::getName() {
00028     return name;
00029 }
00031 void Playlist::setName(string _name) {
00032     name = _name;
00033 }
00035 Playlist Playlist::operator+ (Playlist& segPlaylist) {
00036     Playlist resultante;
00037     resultante.insertSong(*this);
00038     resultante.insertSong(segPlaylist);
00039     return resultante;
00040 }
00042 Playlist::Playlist(const Playlist& old) {
00043     songs = new LinkedList;
00044     playing = nullptr;
00045     count = 1;
00046     name = old.name;
00047     no* temp = old.songs->getHead();
00048     while (temp != nullptr) {
00049         songs->insertEnd(temp->data);
00050         temp = temp->next;
00051     }
00052 }
00054 Playlist Playlist::operator+ (Song& toAdd) {
00055     Playlist resultante;
00056     resultante.insertSong(*this);
00057     resultante.songs->insertEnd(toAdd);
00058     return resultante;
00059 }
00060
00066 Playlist Playlist::operator- (Playlist& segPlaylist) {
00067     Playlist resultante(*this);
00068     no* temp = segPlaylist.getSongs()->getHead();
00070     while (temp != nullptr) {
00072         size_t pos = resultante.getSongs()->getPosition(temp->data);
00073         if (pos > 0) {
00074             resultante.removeSong(pos);
00075         }
00076         temp = temp->next;
00076     }
```

```

00077     }
00078     return resultante;
00079 }
00085 Playlist Playlist::operator- (Song& toRemove) {
00086     Playlist resultante(*this);
00087     size_t pos = resultante.getSongs()->getPosition(toRemove);
00089     if (pos > 0) {
00090         resultante.removeSong(pos);
00091     }
00092     return resultante;
00093 }
00098 void Playlist::operator» (Song*& lastSong) {
00099     if (songs->getSize() > 0) {
00100         no* temp = new no;
00101         *songs » temp;
00102         *lastSong = temp->data;
00103         delete temp;
00104     } else {
00105         lastSong = nullptr;
00106     }
00107 }
00112 void Playlist::operator« (Song*& newSong) {
00113     if (newSong == nullptr) {
00114         return;
00115     } else {
00116         songs->insertEnd(*newSong);
00117     }
00118 }
00124 void Playlist::insertSong(size_t pos, Song value) {
00125     songs->insertPosition(pos, value);
00126     playing = songs->getHead();
00127 }
00128
00133 void Playlist::removeSong(size_t pos) {
00134     songs->removePosition(pos);
00135 }
00136
00142 void Playlist::moveSong(size_t start, size_t end) {
00143     if (start != end) {
00145         no* target = songs->getno(start);
00146         Song value = target->data;
00148         songs->removePosition(start);
00150         if (end < start) {
00151             songs->insertPosition(end, value);
00152         } else {
00153             songs->insertPosition(end + 1, value);
00154         }
00155         playing = songs->getHead();
00156     }
00157 }
00158
00163 void Playlist::insertSong(Playlist& toInsert) {
00165     if (toInsert.getSongs()->getSize() < 1) {
00166         return;
00167     } else {
00168         songs->insertEnd(*toInsert.getSongs());
00169     }
00170 }
00171
00177 size_t Playlist::removeSong(Playlist& toRemove) {
00179     if (toRemove.getSongs()->getSize() < 1) {
00180         return 0;
00181     } else {
00182         size_t removed = 0;
00183         no* temp = toRemove.getSongs()->getHead();
00184         while (temp != nullptr) {
00185             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
00201 void Playlist::displayAllSongs(no* current) {
00203     if (current == nullptr) {
00204         count = 1;
00205         return;
00206     }
00207     cout << count << " - " << current->data.getTitle() << " - " << current->data.getArtist() << endl;
00208     ++count;
00209     displayAllSongs(current->next);
00210 }
00211

```

```
00217 void Playlist::displayOne(no* current, int pos){
00218     if (current == nullptr) {
00219         count = 1;
00220         return;
00221     }
00222     if(count == pos){
00223 cout << current->data.getTitle() << " - " << current->data.getArtist() << endl;
00224     }
00225     ++count;
00226     displayOne(current->next, pos);
00227 }
```

## 5.22 Referência do Arquivo src/utilitarios.cpp

Função que transforma caracteres maiúsculos em minúsculos.

```
#include <iostream>
#include <cctype>
#include "utilitarios.h"
```

Gráfico de dependência de inclusões para utilitarios.cpp:

### Funções

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

### 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 31 do arquivo `utilitarios.cpp`.

```
00031 {
00032     string aux = "-1";
00033     if (isdigit(s[0]) == true) {
00034         return s;
00035     } else {
00036         return aux;
00037     }
00038 }
```

## 5.22.2.2 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 18 do arquivo `utilitarios.cpp`.

```
00018 {
00019     string result = "";
00020     for (size_t i = 0; i < s.size(); i++) {
00021         result += tolower(s[i]);
00022     }
00023     return result;
00024 }
```

## 5.23 utilitarios.cpp

Vá para a documentação desse arquivo.

```
00001
00008 #include <iostream>
00009 #include <cctype>
00010 #include "utilitarios.h"
00011 using namespace std;
00012
00018 string toLowercase(string s) {
00019     string result = "";
00020     for (size_t i = 0; i < s.size(); i++) {
00021         result += tolower(s[i]);
00022     }
```

```
00023     return result;
00024 }
00025
00031 string checkInt(string s){
00032     string aux = "-1";
00033     if (isdigit(s[0]) == true) {
00034         return s;
00035     } else {
00036         return aux;
00037     }
00038 }
```