# INF16153 - Programação II - DI/UFES

# Trabalho Prático I - SpotFES

Prof. Filipe Multz

Prof. Vinicíus Mota

**Data de entrega: 17/11/2022**

**Grupos**: Os trabalhos podem ser realizados em dupla.

**Forma de Entrega:** Os arquivos .h, .c e o Makefile devem ser comprimidos em um arquivo ".zip" e submetidos por todos os integrantes do grupo na atividade do AVA. **Trabalhos entregues fora do prazo definido acima receberão nota zero**.

**Entrevistas:** Haverá uma entrevista que definirá a nota individual de cada aluno.

## 1. Objetivos

O trabalho visa o desenvolvimento das habilidades de criação de structs, gerenciamento de memória alocada dinamicamente, manipulação de arquivos dos tipos texto e binário, organização e documentação correta de códigos-fonte e compilação automática de código.

O trabalho consiste no desenvolvimento de um sistema para recomendação de músicas utilizando uma base de dados extraída de um serviço de streaming. A base contém mais de 600.000 músicas, de 1921 a 2020, mais de 1Mi de artistas e diversas métricas de popularidade das músicas.

## 2. Funcionalidades

Ao ser executado, o programa deve ler os dados de músicas e artistas de dois arquivos do tipo *Comma-Separated Values* (CSV). Os arquivos de entrada serão subconjuntos da base de dados disponível em [1]. O significado das colunas dos arquivos podem ser encontradas em [2]. Os nomes dos arquivos CSV deverão ser recebidos como argumentos na linha de comando.

Deverão ser criadas estruturas Artista e Música para armazenar os dados de uma música e um artista específicos. A estrutura Música deverá conter um array alocado dinamicamente para armazenar os Artistas que produziram a música. As estruturas devem permitir armazenar todos os atributos (colunas) existentes nos arquivos CSV.

O conjunto de todos os dados dos arquivos deverão ser lidos e adicionados à arrays alocados dinamicamente cuja área alocada deverá crescer por demanda. Por exemplo, você pode alocar espaço para 100 itens e sempre que for necessário a área deverá ser realocada para o dobro do tamanho anterior (100 para 200, 200 para 400, 400 para 800 e assim por diante). A decisão do valor inicial faz parte do trabalho.

O programa deve ter um menu com as seguintes opções:

1. **Buscar músicas**: Solicita que o usuário digite um texto e exibe na tela todas as músicas que possuem aquele texto no título, uma por linha. Para cada música, devem ser informados o índice da música no vetor, o id da música (e.g., "1HXdv1z9RlvrcUernyf0MY"), o título da música e os nomes dos artistas.
2. **Listar uma música**: Solicita que o usuário digite o índice de uma música no vetor e exibe na tela todos os atributos da música e dos artistas que a produziram.

   2.1 **Executar uma música:** Faz uma chamada de sistema para abrir a musica no spotify no navegador:

   ```
   firefox https://open.spotify.com/track/0hKRSZhUGEhKU6aNSPBACZ
   ```

3. **Criar uma playlist**: Solicita o nome de uma playlist e a adiciona em um array.
4. **Listar playlists**: Exibe os dados de todas as playlists, uma por linha. Para cada playlist, deve ser exibido o índice da playlist no array, o nome da playlist e o número de músicas que ela possui.
5. **Listar uma playlist**: Solicita que o usuário digite o índice da playlist e apresenta na tela o nome da playlist e os títulos das músicas que ela possui.
6. **Adicionar uma música na playlist**: Solicita o índice de uma música e de uma playlist e adiciona a música a playlist.
7. **Recomendar músicas parecidas com uma playlist**: Solicita o índice de uma playlist e um número inteiro K e mostra na tela as K músicas mais parecidas com àquelas presentes na playlist. Para compreender como será feita a busca por músicas similares, veja a seção "Algoritmo de Recomendação" mais abaixo.
8. **Gerar Relatório**: Salva dois arquivos texto, um contendo as músicas que foram adicionadas em playlists ordenadas de forma decrescente pelo número de playlists às quais elas foram adicionadas, e outro contendo os artistas cujas músicas estão em playlists também ordenados pelo número de vezes que suas músicas foram adicionadas em playlists.

Para representar cada playlist deve ser criada uma estrutura que armazene o nome da playlist e um vetor de inteiros com os índices das músicas que foram adicionadas na playlist. Como no caso das músicas, o conjunto de playlists deve ser armazenado em um vetor alocado dinamicamente e realocado quando necessário.

Ao encerrar o programa, as playlists devem ser salvas em um arquivo binário. Ao iniciar novamente o programa, as playlists devem ser carregadas de forma que o programa sempre tenha acesso ao registro de todas as playlists que já foram cadastradas.

O programa não deve assumir que o número de músicas, artistas ou playlists é conhecido. Para teste do programa serão providos arquivos com diferentes tamanhos e o programa deverá funcionar em todos os casos.

## 3. Algoritmo de Recomendação

No arquivo CSV, para cada música são dadas informações como o quão dançantes são as músicas (danceability), o quão energéticas elas são (energy), o volume (loudness), entre outras. Para a criação do

sistema de recomendação serão consideradas as características:

- danceability
- energy
- mode
- speechiness
- acousticness
- instrumentalness
- liveness
- valence

O sistema de recomendação deverá buscar músicas que tenham características similares àquelas encontradas em uma dada playlist. Como a playlist pode conter músicas com diferentes estilos, será necessário considerar as características que são mais comuns/frequentes. Para isto, deve ser calculada a média dos valores das características para todas as músicas na playlist. Estes valores devem ser armazenados em um array.

Em seguida, devem ser buscadas as K músicas com características mais similares à média da playlist. A similaridade entre as características será dada pela distância euclidiana:

$$d(x, y) = \sqrt{\sum_{k=1}^{n} (x_k - y_k)^2}$$

onde x é o vetor de características da música e y é o vetor com os valores médios da playlist. Quanto menor a distância euclidiana entre dois vetores, mais próximos eles são no espaço e portanto, mais similares.

Dado um número inteiro K, o sistema de recomendação deve buscar as K músicas com menor distância euclidiana em relação ao vetor da playlist. Uma forma de alcançar este objetivo é calcular a distância para todas as músicas, ordenar as músicas de forma crescente por suas distâncias e selecionar as K primeiras. Existem formas mais eficientes de realizar esta busca e quem descobrir poderá utilizá-las 😃

## 4. Conjunto de Dados

Os arquivos estão como *Comma-Separated Values* (CSV) dentro do diretório data.

Os arquivos `artists_full.csv` e `tracks_full.csv` contém todas as informações que devem ser armazenadas em estruturas dos artistas e músicas respectivamente.

Para cada um destes arquivos, estão sendo disponibilizado vários arquivos que são subconjuntos do arquivo original. Nestes casos, os arquivos são nomeados `artists_N.csv`e `tracks_N.csv` contém as musicas de um subconjunto de `N` artistas.

### Artistas

Os arquivo de artistas possuem colunas:

- id: id do artista na base do Spotify;
- Seguidores: Número de seguidores;
- Generos: Uma lista de generos associados ao artista. Observe que por ser uma lista, deverá tratar diferentemente;
- Nome do artista;
- Popularidade: Popularidade do artista entre 0 e 100.

As colunas são separadas por ponto-e-vírgula ';' sendo que o gênero pode conter zero (representado por um hifen '-') ou mais valores, neste caso separados por um pipeline '|'.

Exemplo de artistas no arquivo:

```
0k17h0D3J5VfsdmQ1iZtE9;14343532.0;album rock|art rock|classic
rock|progressive rock|psychedelic rock|rock|symphonic rock;Pink Floyd;83

3WrFJ7ztbogyGnTHbHJFl2;19214899.0;beatlesque|british invasion|classic
rock|merseybeat|psychedelic rock|rock;The Beatles;88


3V4ODgUObsEHFX6GQUy8xl;419.0;deep comedy;Matt McCarthy;9

0DotfDlYMGqkbzfBhcA5r6;7.0;;Astral Affect;0
```

## Músicas (tracks)

Cada linha no arquivo de tracks contém 22 colunas separadas por ';':

- id: id spotify da track
- nome: nome da música
- popularity: Popularidade da música entre 0 e 100
- duracao_ms: Duração da música em ms
- explicit: Se contem conteudo explicito ou não
- artists: Listas de artistas que criaram a musica
- id_artists: id dos artistas que criaram a música
- data de lançamento

Seguido de um conjunto de propriedades listadas abaixo (mantendo o nome dado pelo spotify):

- danceability: [0, 1]
- energy: [0, 1]
- key: nota maior [0: C, 1: C#/Db, 2: D, …]
- loudness: em db
- mode: 0 se minor e 1 se major
- speechiness: [0,1]
- acousticness: [0,1]
- instrumentalness: [0,1]
- liveness:[0,1]

- valence: [0,1]
- tempo: BPM
- time_signature

A explicação de cada propriedade pode ser obtida em [2].

As colunas são separadas por ponto-e-vírgula ';' sendo que o artists e id_atists podem conter um ou mais valores, neste caso separados por um pipeline '|'. Segue exemplo de duas entradas de músicas.

```
45XJsGpFTyzbzeWK8VzR8S;A Day At A Time;58;142003;0;Gentle Bones|Clara
Benin;4jGPdu95icCKVF31CcFKbS|5ebPSE9YI5aLeZ1Z2gkqjn;2021-03-
05;0.696;0.615;10;-6.212;1;0.0345;0.206;2.53e-06;0.305;0.438;90.029;4

27Y1N4Q4U3EfDU5Ubw8ws2;What They'll Say About
Us;70;187601;0;FINNEAS;37M5pPGs6V1fchFJSgCguX;2020-09-
02;0.535;0.314;7;-12.823;0;0.0408;0.895;0.00015;0.0874;0.0663;145.095;4
```

## 5. Avaliação

O trabalho será corrigido em dois turnos: correção do professor e entrevista. A correção do professor (CP) gerará uma nota entre 0 e 10 e as entrevistas (E) serão realizadas posteriormente à data de entrega do trabalho e será atribuída uma nota entre 0 e 1. Alunos que fizeram o trabalho em dupla farão a entrevista juntos, porém terão seus desempenhos na entrevista avaliados de forma individual. Pequenas e pontuais correções de código serão permitidas no momento da entrevista, o que poderá acarretar uma CP maior.

A nota final (NF) do trabalho será dada pela seguinte fórmula: NF = E * CP

O trabalho será pontuado de acordo com sua funcionalidade e outros aspectos de implementação, conforme as listas abaixo.

## 6. Critérios de Qualidade

**Entrega do Trabalho**

- Para cada estrutura devem ser criados arquivos .h e .c contendo a definição da estrutura e funções relacionadas àquele tipo de dado. Estes arquivos devem ser armazenados em um subdiretório. Deve ser criada uma biblioteca para armazenar os arquivos objeto (.o) relativos a todos os arquivos .c exceto o arquivo que contém a função main. Com esta estrutura, o programa principal deve ser compilável usando uma instrução como:

```
gcc -o main main.c -L . -lrecomendacao -I ./recomendacao/
```

- Deve ser provido um Makefile com regras para compilar e executar o programa, e para limpar o projeto. O makefile deve ser construído de forma a só recompilar aquilo que for necessário dadas as modificações no código (e.g., se apenas o arquivo main.c for modificado, a biblioteca não precisa ser reconstruída).

- Trechos de códigos que não sejam totalmente claros devem ser comentados.
- Variáveis e funções devem possuir nomes que tornem o código legível e compreensível sem a necessidade de comentários frequentes.
- Segmentation Fault, erros no valgrind e alocação de memória além do desnecessário, levarão a perdas substanciais de pontuação. Trabalhos que alocarem grades quantidades de memória para tentar evitar o uso do mecanismo de realocação receberão nota zero.

**Aspectos funcionais**

1. (10%) Inicialização. Capacidade de carregar os arquivos usando estruturas dinâmicas.
2. (5% cada) Funcionalidades 1 a 6 descritas na Seção 2.
3. (10%) Recomendar músicas parecidas com uma playlist (Funcionalidade 7).
4. (10%) Gerar Relatório, salvando e carregando os arquivos de playlists corretamente.

**Aspectos de desenvolvimento:**

1. (10%) Uso correto de ponteiros e alocação de memória. O uso incorreto de declaração estática de vetores será penalizado em até 100% da nota.
2. (10%) Uso correto de TADs para abstração dos elementos envolvidos no trabalho. Menos *main* mais TADs! (20%)
3. (5%) Código legível. Variáveis com nomes claros, usar constantes ao invés de números mágicos.
4. (5%) Qualidade da documentação. Fazer comentários que indiquem claramente, como o código funciona e como usar as funções implementadas.
5. (10%) Executar o programa com o valgrind, sair corretamente e não ter vazamento de memória.

# Referências

[1] https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks

[2] https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features