

# PlayED

---

por *Álvaro Davi S. Alves* - 2020101874

Primeiro trabalho de Estrutura de Dados I

Turma de Engenharia da Computação

Universidade Federal do Espírito Santo ([UFES](#))



- 
- [x] Código criado com conteúdo visto em aula
    - [x] Tipos Abstratos de Dados
    - [x] Estruturas Opacas
    - [x] Ponteiros para Funções
    - [x] Listas Encadeadas
      - [x] Simplesmente Ligadas
      - [x] Duplamente Ligadas
    - [x] Gerenciamento de Memória
    - [x] Modularização
    - [x] Leitura de Arquivos
  - [x] Documentar funções, estruturas, constantes e macros nos arquivos header em `include` (comentários multilinha)
  - [x] Documentar trechos complicados do código nos arquivos source em `source` (comentários em uma linha)
  - [x] Criar makefile para compilar os arquivos
- 

## Introdução:

---

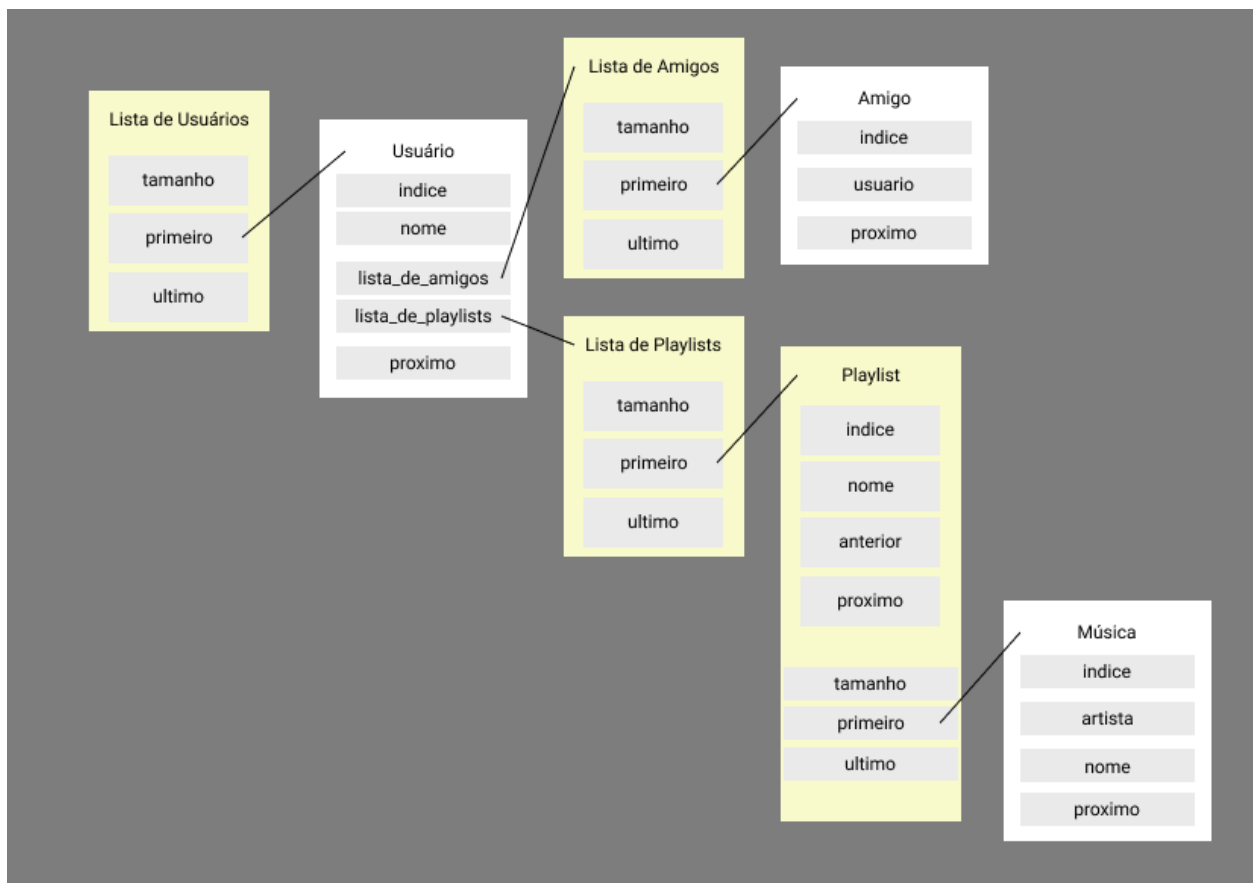
Este trabalho visa a criação de um programa para gerenciar músicas em playlists e relação de usuários no estilo rede social escrito na linguagem procedural estruturada C utilizando tipos abstratos de dados, modularização, leitura de arquivos, estruturas de dados e gerenciamento de memória.

## Implementação:

---

O programa é dividido nos arquivos `friendship`, `playlists`, `users` e `utils`, onde a implementação das funções e dos tipos abstratos se encontram em arquivos de extensão `.c` no diretório `_source_` e suas definições e protótipos se encontram em arquivos de extensão `.h` no diretório `_include_`.

Abaixo, o diagrama representativo de cada tipo implementado:



A seguir, alguns exemplos da declaração das funções e sua documentação dentro do código, de forma que fiquem compreensíveis e bem documentadas mesmo utilizando ferramentas de IntelliSense existentes na maioria dos editores de código e IDEs.

```
main.c - PlayED-Data_Structures - Visual Studio Code

x  README.md M  C  utils.c

C main.c > main()
// opening base files
FILE *friendship_if = fopen("./input/amizade.txt", "r"); // friendship input file
FILE *playlists_if = fopen("./input/playlists.txt", "r"); // playlists input file

if (friendship_if == NULL || playlists_if == NULL) // check if the files do not exist
{
    int createFolder(char *path, char *dirName) // create folder to save playlists.txt files, exiting!\n");
    Create specific folder to save output files

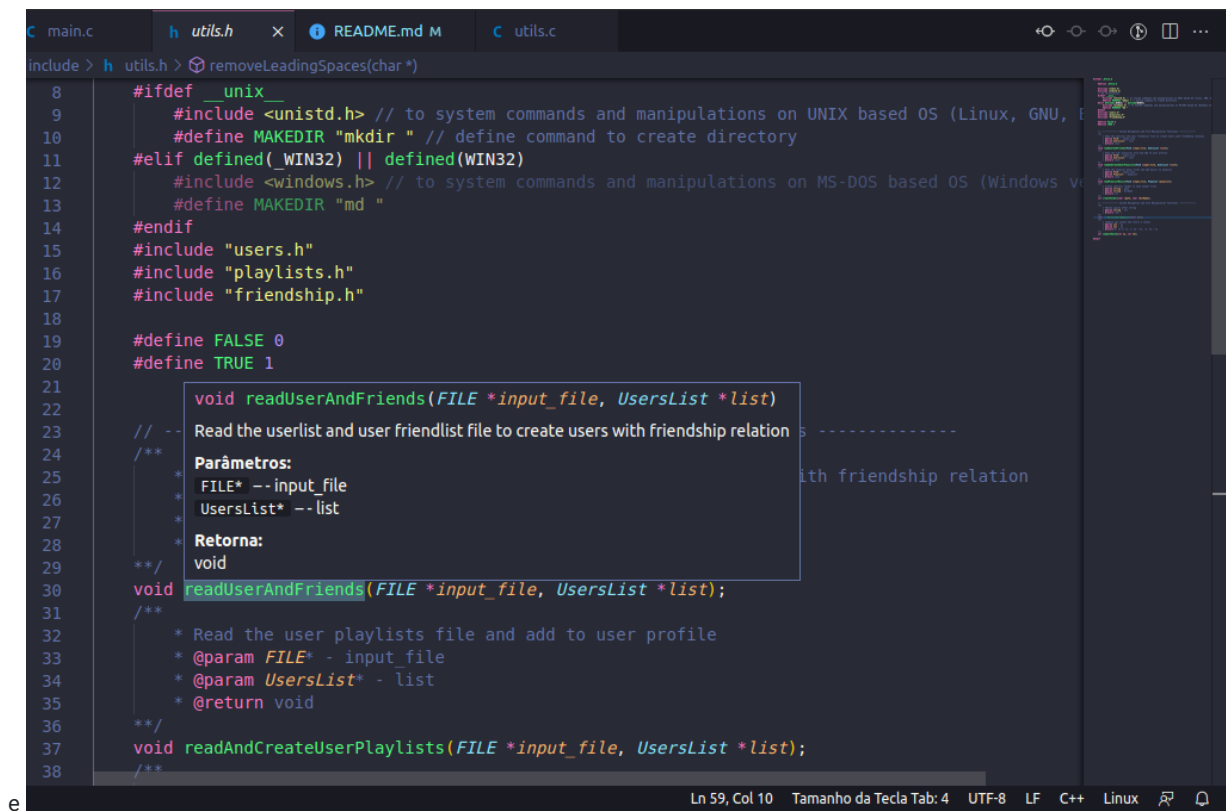
    Parâmetros:
    string --path
    string -- dirName

    Retorna:
    int

    createFolder("./", "output");
    refactPlayED(users);
    printSimilarities(users);
    You, 14 hours ago via PR #5 * refactored plED: output file following the pattern
}

/**
 * TODO: URGENT!!!
 * ? fix memory leak in readUserAndFriends on utils.c
 * * create similarities function
 */

// freeing memory
destroyUsersList(users);
system("rm -rf ./temp"); // clear all temporary files
```



```
1  main.c  h  utils.h  x  README.md M  C  utils.c
include > h  utils.h  removeLeadingSpaces(char *)
8  #ifndef __unix__
9  #include <unistd.h> // to system commands and manipulations on UNIX based OS (Linux, GNU, BSD, etc)
10 #define MAKEDIR "mkdir " // define command to create directory
11 #elif defined(_WIN32) || defined(WIN32)
12 #include <windows.h> // to system commands and manipulations on MS-DOS based OS (Windows, etc)
13 #define MAKEDIR "md "
14 #endif
15 #include "users.h"
16 #include "playlists.h"
17 #include "friendship.h"
18
19 #define FALSE 0
20 #define TRUE 1
21
22 void readUserAndFriends(FILE *input_file, UsersList *list)
23 // -- Read the userlist and user friendlist file to create users with friendship relation
24 /**
25  * Parâmetros:
26  * FILE* --input_file
27  * UsersList* --list
28  * Retorna:
29  * void
30  */
31 void readUserAndFriends(FILE *input_file, UsersList *list);
32 /**
33  * Read the user playlists file and add to user profile
34  * @param FILE* - input_file
35  * @param UsersList* - list
36  * @return void
37  */
38 void readAndCreateUserPlaylists(FILE *input_file, UsersList *list);
39 /**
```

As principais funções existentes no programa podem explicadas em linguagem natural como:

```
UsersList * initUsersList() - inicia lista de usuarios vazia
User * registerUser(char *name) - cria um novo usuário com o nome fornecido

FriendList * initFriendList() - inicia uma lista de amigos vazia
Friend * makeFriend(User *usr) - cria um novo elemento para a lista de amigos que aponta para um usuário em específico

PlaylistList * initPlaylistList() - inicia uma lista de playlists vazia
Playlist * initPlaylist(char *name) - inicia uma playlist vazia que é um elemento de uma lista e uma lista ao mesmo tempo
Music * createMusic(char *name, char *artist) - cria uma música com nome e artista fornecidos
```

- **Estruturas e TADs Implementados:**  
A escolha das estruturas de cada TAD foi feita após a análise das funcionalidades pedidas, visando facilidade, clareza e versatilidade para o programa.
  - **Lista de Usuários:** lista individualmente encadeada contendo Tamanho, Primeiro e Último Usuário
  - **Usuário:** TAD contendo Índice, Nome, Lista de Amigos, Lista de Playlists e Próximo Usuário
  - **Lista de Amigos:** lista individualmente encadeada contendo Tamanho, Primeiro e Último Amigo
  - **Amigo:** TAD contendo Índice, Usuário e Próximo Amigo
  - **Lista de Playlists:** lista duplamente encadeada contendo Tamanho, Primeira e Última Playlist
  - **Playlist:** TAD contendo Índice, Nome, Anterior e Próxima Playlist e lista individualmente encadeada contendo Tamanho, Primeira e Última Música >
  - **Música:** TAD contendo Índice, Nome, Artista e Próxima Música

## Conclusão:

O trabalho abordou de forma clara e eficiente o uso de estruturas de dados para os mais diversos fins, explicitando a aplicação de tais estruturas desde os mais simples programas aos mais complexos sistemas.

As maiores dificuldades encontradas foram no uso dos TADs de forma entrelaçada, no gerenciamento de memória e em formas eficientes de criar algoritmos.

## Bibliografia:

[listas encadeadas - IME-USP](#)  
[singly-linked list - TutorialsPoint](#)  
[doubly-linked list - GeeksforGeeks](#)