UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DA COMPUTAÇÃO

GABRIELA ANDRADE MARTINS CUNHA
GUILHERME SILVA COTRIM

Trabalho Final:
Netflix

Uberlândia
2025

GABRIELA ANDRADE MARTINS CUNHA
GUILHERME SILVA COTRIM

Trabalho Final:
Netflix

Trabalho final da disciplina de
Algoritmos e Estruturas de Dados I

Uberlândia
2025

SUMÁRIO

1 INTRODUÇÃO

Neste projeto, desenvolvemos uma solução para criação de perfis de usuários em que cada usuário possui uma lista de séries assistidas. A estrutura consiste em quatro arquivos principais: dois arquivos de cabeçalho (series.h e perfis.h), que contêm os protótipos das funções relacionadas e tipos de dados armazenados, e dois arquivos (series.c e perfis.c) onde é feita a implementação das funções declaradas nos respectivos cabeçalhos. No main.c ocorre a lógica principal do programa, onde ocorrem a criação dos perfis, registro das séries assistidas e a manipulação das listas duplamente ligadas.

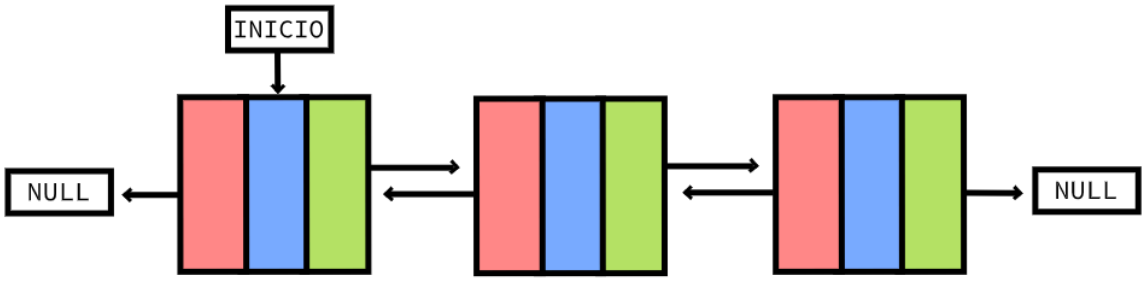
A solução foi planejada com base na modularização do problema, separando os arquivos facilitando a manutenção e organização do código. O uso de listas duplamente encadeadas, como pedido no próprio enunciado do problema, permitiu a navegação eficiente e flexível onde cada nó aponta para o seu antecessor e sucessor. Além disso, o uso de um nó descritor contribui para o gerenciamento da lista, armazenando os ponteiros de início e fim.

1.1 Ilustrações da Estruturas das Listas Utilizadas

Imagem 1 – Informações do nó da lista

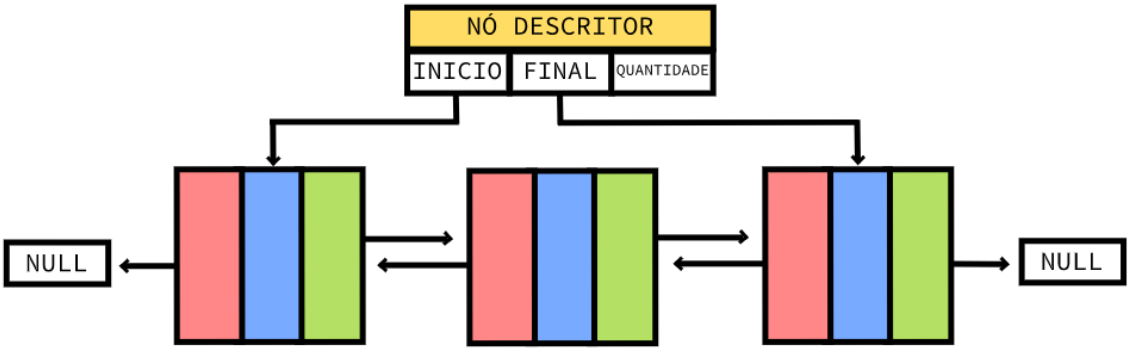
Fonte: Autoria própria.

Imagem 2 – Lista duplamente encadeada



Fonte: Autoria própria.

Imagem 3 – Lista duplamente encadeada com nó descritor



Fonte: Autoria própria.

2 SOBRE O CÓDIGO

2.1 Funções series.c

2.1.1 Carregar séries (loadTVShows)

A função, em seu aspecto geral, carrega os dados dos arquivos e preenche a lista com esses dados.

a) Verifica se o ponteiro *pn* é nulo, caso sim retorna 0.
b) Monta um caminho para o arquivo e abre o arquivo.
c) Lê cada linha do arquivo.
d) Usa o *strtok* para dividir os campos do arquivo.
e) Preenche a *struct* com os dados extraídos.
f) Aplica *trim* (função do perfis.h) para remover os caracteres indesejados.
g) Adiciona a série a estrutura.
h) Fecha o arquivo

```
//Arquivo series.h
int loadTVShows(Prof_Node *pn, const char *filename);

//Arquivo series.c
int loadTVShows(Prof_Node *pn, const char *filename) {
    if (pn == NULL) return 0;

    char str1[11];
    strcpy(str1, "archives/"); // melhorar isso depois
    strcat(str1,filename);

    FILE* file = fopen(str1, "r");
    if (file == NULL) {
        //printf("Erro ao abrir o arquivo");
        return 0;
    }

    TVShow t;
    char linha[200];
    while (fgets(linha, sizeof(linha), file)) {
        linha[strcspn(linha, "\n")] = '\0';

        strcpy(t.name, strtok(linha, "|"));
        strcpy(t.broadcaster, strtok(NULL, "|"));
        strcpy(t.creator, strtok(NULL, "|"));
        strcpy(t.genre, strtok(NULL, "|"));
        t.seasons = strtol(strtok(NULL, "|"),NULL,10);
        t.episodes = strtol(strtok(NULL, "|"),NULL,10);
```

```
        t.year = strtol(strtok(NULL, "|"),NULL,10);
        strcpy(t.ratingIMDB, strtok(NULL, "|"));
        t.personalRating = strtol(strtok(NULL, "|"),NULL,10);

        char *temp = strtok(NULL, ";");
        trim(temp);
        t.favorite = temp[0];

        trim(t.name);
        trim(t.broadcaster);
        trim(t.creator);
        trim(t.genre);
        trim(t.ratingIMDB);

        addTVShow(pn, t);
    }
    fclose(file);
    return 1;
}
```

2.1.2 Adiciona séries (addTVShow)

É uma função clássica de inserção.

.

a) Verifica se *pn* está vazio.
b) Aloca um novo nó em *TVS_Node* para armazenar a série.
c) Caso a lista esteja vazia define como a primeira série da lista definindo como primeiro e o último.
d) Se houver elementos ela adiciona no final, atualizando *next* e o *before.*
e) Incrementa o contador de séries.

```
//Arquivo series.h
int addTVShow(Prof_Node *pn, TVShow tvs);

//Arquivo series.c
int addTVShow(Prof_Node *pn, TVShow tvs) {
    if (pn == NULL) return 0;
    TVS_Node *newNode = malloc(sizeof(TVS_Node));
    if (newNode == NULL) return 0; // erro de alocação

    newNode->info = tvs;
    if (pn->start == NULL && pn->end == NULL) { // primeira série da lista
        newNode->next = NULL;
        newNode->before = NULL;
        pn->start = newNode;
        pn->end = newNode;
        pn->quantTVShows++;
        return 1;
    }
    newNode->next = NULL;
    newNode->before = pn->end;
```

```
    pn->end->next = newNode;
    pn->end = newNode;
    pn->quantTVShows++;
    return 1;
}
```

2.1.3 Imprime séries (printTVShows)

Função de impressão da lista de acordo usuário.

a) Verifica se a lista de perfis está vazia.
b) Começa a imprimir de acordo com usuário, percorrendo a lista com *aux* e indo para o próximo nó com *aux* = *aux->next.*

```
//Arquivo series.h
void printTVShows(Prof_Node *pn);

//Arquivo series.c
void printTVShows(Prof_Node *pn) {
    if (pn == NULL) {
        printf("Lista vazia!");
        return;
    }
    if (pn->quantTVShows == 0) {
        printf("Nenhuma serie foi adicionada ainda.\n");
        return;
    }

    printf("Series de %s:\n",pn->info.name);
    TVS_Node *aux = pn->start;
    while (aux != NULL) {
        //if (aux->info.favorite == 'Y')
        if (aux->info.favorite == 'Y' || aux->info.favorite == 'y') {
            printf("\nS2 %s (%d)\n",aux->info.name,aux->info.year);
        }else {
            printf("\n%s (%d)\n",aux->info.name,aux->info.year);
        }
        printf("Emissora: %s | Criador(es):
%s\n",aux->info.broadcaster,aux->info.creator);
        printf("No de Temporadas: %d | No de Episodios:
%d\n",aux->info.seasons, aux->info.episodes);
        printf("Categoria: %s\n",aux->info.genre);
        printf("Nota IMDB: %s | Nota Pessoal: %d\n",aux->info.ratingIMDB,
aux->info.personalRating);
        printf("-----------------------------------\n\n");
        aux = aux->next;
    }
}
```

2.1.4 Remoção de série (removeTVShow)

Função de remoção.

a) Verifica se a lista existe.
b) Percorre a lista com *aux*, procurando o parâmetro igual a *name*
c) Se não encontrar retorna -1 (série não está no perfil).
d) Se encontrar e for a primeira, atualiza *pn->start.*
e) Se for a última, atualiza *pn->end.*
f) Se estiver no meio ajusta *next* e *before.*
g) Libera memória do que foi removido

```
//Arquivo series.h
int removeTVShow(Prof_Node *pn, const char *name);

//Arquivo series.c
int removeTVShow(Prof_Node *pn, const char *name) {
    if (pn == NULL) return 0;

    TVS_Node *aux = pn->start;
    while (aux != NULL) {
        if (strcmp(aux->info.name, name) == 0) {
            break;
        }
    }
    if (aux == NULL) return -1; // serie nao esta no perfil;

    if (aux == pn->start) { // série é a primeira na lista
        pn->start = aux->next;
        pn->start->before = NULL;
        pn->quantTVShows--;
        free(aux);
        return 1;
    }
    if (aux == pn->end) { // série é a ultima na lista
        pn->end = aux->before;
        pn->end->next = NULL;
        pn->quantTVShows--;
        free(aux);
        return 1;
    }
    // série é esta no meio na lista
    aux->next->before = aux->before;
    aux->before->next = aux->next;
    pn->quantTVShows--;
    free(aux);
    return 1;
}
```

2.1.5 Pesquisar série (searchTVShow)

Procura a série pelo nome dentro da lista de séries de um perfil.

a) Verifica se o perfil existe.
b) Percorre a lista comparando o nome de cada série.
c) Se encontrar sai do loop e retorna o ponteiro para o nó da série.
d) Se não encontrar continua até o final da lista.
e) Se no final não encontrou retorna NULL.

```
//Arquivo series.h
TVS_Node* searchTVShow(Prof_Node *pn, const char *name);

//Arquivo series.c
TVS_Node* searchTVShow(Prof_Node *pn, const char *name) {
    if (pn == NULL) return NULL;
    TVS_Node *aux = pn->start;
    while (aux != NULL) {
        if (strcmp(aux->info.name, name) == 0) {
            break;
        }
        aux = aux->next;
    }
    if (aux == NULL) return NULL;
    return aux; // retorna o nó
}
```

2.1.6 Imprimir os favoritos (printFavorites)

Filtra e exibe somente as séries marcadas como favoritas.

a) Verifica se a lista de perfis existe.
b) Percorre a lista.
c) Verifica se a série tem o campo *favorite* marcado como sim.
d) Imprime somente as séries com campo favorito marcado como sim.

```
//Arquivo series.h
void printFavorites(Prof_Node *pn);

//Arquivo series.c
void printFavorites(Prof_Node *pn) {
    if (pn == NULL) return;
    int cont = 0;
    TVS_Node *aux = pn->start;
    printf("--- Series Favoritas de %s---\n",pn->info.name);
    while (aux != NULL) {
        if (aux->info.favorite == 'Y' || aux->info.favorite == 'y') {
            printTVShows(pn, aux);
            cont++;
        }
```

```
        aux = aux->next;
    }
    printf("\nSeries Favoritas: %d\n",cont);
}
```

2.1.7 Favoritar série (favoriteTVShow)

Adiciona uma série aos favoritos.

a) Verifica se a lista de perfis existe.
b) Percorre a lista.
c) Verifica se a série tem o campo *favorite* marcado como não.
d) Se marcado como não, muda a informação, trocando o status para favorito.
e) Se o campo sim estiver marcado, já era favorito.

```
//Arquivo series.h
int favoriteTVShow(Prof_Node *pn, const char *name);

//Arquivo series.c
int favoriteTVShow(Prof_Node *pn, const char *name) {
    if (pn == NULL) return 0;

    TVS_Node *aux = pn->start;
    while (aux != NULL) {
        if (strcmp(aux->info.name, name) == 0) {
            if (aux->info.favorite == 'N' || aux->info.favorite == 'n') {
                aux->info.favorite = 'Y'; // remove o status de favorito
                return 1;
            }
            else {
                return -1; // ja era favorito;
            }
        }
        aux = aux->next;
    }
    return 0; // não encontrada;
}
```

2.1.8 Remover série favorita (unfavoriteTVShow)

A função remove a série favorita.

a) Verifica se a lista de perfis existe.
b) Percorre a lista.
c) Verifica se a série tem o campo *favorite* marcado como sim.
d) Se marcado como sim, muda a informação, trocando o status de favorito.
e) Se o campo não estiver marcado, não era favorito.

```
//Arquivo series.h
int unfavoriteTVShow(Prof_Node *pn, const char *name);

//Arquivo series.c
int unfavoriteTVShow(Prof_Node *pn, const char *name) {
    if (pn == NULL) return 0;

     TVS_Node *aux = pn->start;
     while (aux != NULL) {
        if (strcmp(aux->info.name, name) == 0) {
            if (aux->info.favorite == 'Y' || aux->info.favorite == 'y') {
                aux->info.favorite = 'N'; // remove o status de favorito
                return 1;
            } else {
                return -1; // não era favorito
            }
        }
        aux = aux->next;
    }

    return 0; // não encontrada
}
```

2.1.9 Libera lista de séries (freeTVShows)

Libera lista de séries (usada com freeList()).

a) Verifica se a lista não está vazia.
b) Declara um ponteiro auxiliar para ajudar na liberação dos nós.
c) Percorre até o final.
d) Guarda o atual, avança para o próximo e libera o nó anterior.

```
//Arquivo series.h
void freeShows(TVS_Node *l);

//Arquivo series.c
void freeShows(TVS_Node *l) { // libera lista de séries
    if (l != NULL) {
        TVS_Node *aux;
        while (l != NULL) {
            aux = l;
            l = l->next;
            free(aux);
        }
    }
}
```

2.2 Funções perfis.c

2.2.1 Criar lista de perfil (createList)

Inicializa a estrutura principal que vai armazenar os perfis.

a) Aloca a quantidade de memória para armazenar *D_profiles.*
b) Inicializa o ponteiro para apontar o primeiro perfil da lista como *NULL*, e logo em seguida, faz a mesma coisa com o último perfil da lista.
c) Define o contador da lista como zero.
d) Retorna a estrutura criada.

```
//Arquivo perfis.h
D_profiles* createList();

//Arquivo perfis.c
D_profiles* createList() {
    D_profiles *li = malloc(sizeof(D_profiles));
    li->start = NULL;
    li->end = NULL;
    li->quantProfiles = 0;
    return li;
}
```

2.2.2 Adiciona perfil (addProfile)

Funciona de forma parecida com *addTVShow*, como uma função de inserção.

a) Verifica se a lista existe.
b) Aloca um novo nó em *Prof_Node* para guardar o perfil.
c) Copia os dados do perfil para o nó.
d) Inicializa a lista de series do perfil como *NULL.*
e) Se a lista de perfil está vazia, se estiver vazia define o começo e o fim da lista incrementa ao contador.
f) Se a lista já possui perfis, conecta o último com o atual, o atual será o novo último, último aponta para o novo, atualiza o fim da fila e incrementa o contador.

```
//Arquivo perfis.h
```

```
int addProfile(D_profiles *li, Profile p);

//Arquivo perfis.c
int addProfile(D_profiles *li, Profile p){
    if (li == NULL) return 0;
    Prof_Node *newProfile = malloc(sizeof(Prof_Node));
    if (newProfile == NULL) return 0; // erro de alocação;

    newProfile->info = p;
    newProfile->start = NULL; // null para lista de séries (que não foi criada
ainda)
    newProfile->end = NULL; // null para lista de séries (que não foi criada
ainda)
    newProfile->quantTVShows = 0;
    if (li->quantProfiles == 0) { // adicionar primeiro perfil
        newProfile->next = NULL;
        newProfile->before = NULL;

        li->start = newProfile;
        li->end = newProfile;
        li->quantProfiles++;
        return 1;
    }
    newProfile->before = li->end; // before do novo nó é o fim
    newProfile->next = NULL; // next do novo nó é nul
    li->end->next = newProfile; // next do fim é o novo nó
    li->end = newProfile; // novo fim é o novo nó
    li->quantProfiles++; // incrementa quantidade de perfis
    return 1;
}
```

2.2.3 Carrega perfis (load_profiles)

De forma parecida com *loadTVShows*, carrega os dados dos arquivos e preenche a lista com esses dados.

a) Verifica se a lista é vazia.
b) Abre o arquivo, se falhar retorna “Erro ao abrir arquivo”.
c) Lê cada linha do arquivo.
d) Divide o arquivo usando “|” e “;” como delimitador.
e) Preenche os campos.
f) Usa *trim* para remover os espaços desnecessários.
g) Insere o perfil na lista.
h) Fecha o arquivo.

```
//Arquivo perfis.h
int load_profiles(D_profiles *li, const char *filename);
```

```
//Arquivo perfis.c
int load_profiles(D_profiles *li, const char *filename) {
    if (li == NULL) return 0;
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("Erro ao abrir o arquivo");
        return 0;
    }

    Profile p;
    char linha[200];
    while (fgets(linha, sizeof(linha), file)) {
        linha[strcspn(linha, "\n")] = '\0';

        strcpy(p.name, strtok(linha, "|"));
        //p.age = atoi(strtok(NULL, ";")); // tentativa que ficava dando
warning
        strcpy(p.age, strtok(NULL, ";"));
        trim(p.name);
        trim(p.age);
        addProfile(li, p);
    }
    fclose(file);
    return 1;
}
```

2.2.4 Impressão dos perfis (printProfiles)

Exibe os perfis cadastrados.

a) Verifica se a lista foi criada.
b) Percorre a lista de perfis.
c) Para cada perfil imprime as respectivas informações.
d) Exibe a quantidade de perfis cadastrados.

```
//Arquivo perfis.h
void printProfiles(D_profiles *li);

//Arquivo perfis.c
void printProfiles(D_profiles *li) {
    if (li == NULL) {
        printf("Lista Vazia!\n");
        return;
    }
    printf("\n----- PERFIS ------\n\n");
    Prof_Node *aux = li->start;
    while (aux != NULL) {
        printf("%s | Idade: %s | Series Assistidas: %d\n", aux->info.name,
aux->info.age, aux->quantTVShows);
        aux = aux->next;
    }
    printf("\n>> Perfis Cadastrados: %d\n", li->quantProfiles);
```

```
    printf("\n----------------------\n");
}
```

2.2.5 Remoção de Perfil (removeProfile)

Remove o perfil de acordo com o nome.

a) Verifica se a lista existe.
b) Percorre a lista procurando o perfil com o nome a ser excluído.
c) Se encontrar sai do loop e aponta para o perfil a ser removido.
d) Se não encontrar, retorna -1, ou seja, perfil não existe.
e) Se for o primeiro, atualiza o começo da lista, remove o perfil a e libera memória.
f) Se for o último, atualiza o fim, remove o perfil e libera memória.
g) Se estiver no meio, conecta o anterior e próximo do nó, remove o nó e libera memória.

```
//Arquivo perfis.h
int removeProfile(D_profiles *li, const char *name);

//Arquivo perfis.c
int removeProfile(D_profiles *li, const char *name) {
    if (li == NULL) return 0;
    Prof_Node *aux = li->start;
    while (aux != NULL) {
        if (strcmp(aux->info.name, name) == 0) {
            break;
        }
        aux = aux->next;
    }
    if (aux == NULL) return -1; // perfil não existe;
    if (aux == li->start) { // perfil a ser removido é o primeiro
        li->start = aux->next;
        li->start->before = NULL;
        li->quantProfiles--;
        free(aux);
        return 1;
    }
    if (aux == li->end) { // perfil a ser removido é o último
        li->end = aux->before;
        li->end->next = NULL;
        li->quantProfiles--;
        free(aux);
        return 1;
    }
    // perfil a ser removido é algum do meio
    aux->next->before = aux->before;
    aux->before->next = aux->next;
    li->quantProfiles--;
```

```
    free(aux);
    return 1;
}
```

2.2.6 Pesquisa Perfil (searchProfile)

Função de busca.

a) Se estiver vazia retorna *NULL*.
b) Cria um ponteiro auxiliar para percorrer a lista do início.
c) Percorre a lista de perfis.
d) Compara os nomes.
e) Se encontrar, sai do loop e retorna o ponteiro para o perfil.
f) Se não encontrou, retorna *NULL*.

```
//Arquivo perfis.h
Prof_Node* searchProfile(D_profiles *li, const char *name);

//Arquivo perfis.c
Prof_Node* searchProfile(D_profiles *li, const char *name) {
    if (li == NULL) return NULL;
    Prof_Node *aux = li->start;
    while (aux != NULL) {
        if (strcmp(aux->info.name, name) == 0) {
            break;
        }
        aux = aux->next;
    }
    if (aux == NULL) return NULL;
    return aux;
}
```

2.2.7 Aparar ou Podar(trim)

Remove espaços em branco de uma string.

a) Avança o ponteiro para encontrar os lugares que não é espaço e contá-los
b) Se a *string* só for espaços, ela é vazia.
c) Vai até o último caractere que não é espaço e adiciona *\0* após o último caractere.
d) Move a parte útil da *string* eliminando os espaços.

```
//Arquivo perfis.h
void trim(char *str);

//Arquivo perfis.c
void trim(char *str) {
    char *start, *end;
    // Encontrar o primeiro caractere que não é espaço em branco
    start = str;
    while (*start && isspace((unsigned char)*start)) {
        start++;
    }
    // Se a string estiver vazia, retorna
    if (*start == 0) {
        *str = '\0';
        return;
    }
    // Encontrar o último caractere que não é espaço
    end = start + strlen(start) - 1;
    while (end > start && isspace((unsigned char)*end)) {
        end--;
    }
    // Adiciona o caractere nulo após o último caractere que não é espaço
    *(end + 1) = '\0';
    // Mover a string para o início da posição de memória original
    if (start != str) {
        memmove(str, start, end - start + 2);
    }
}
```

2.2.8 Editar perfil (changeInfo)

Pergunta novamente nome e idade e altera informações de um perfil específico.

a) Verifica se o ponteiro para o perfil é nulo.
b) Copia o nome novo do perfil para um já existente e faz o mesmo com a idade.

```
//Arquivo perfis.h
int changeInfo(Prof_Node *pn, Profile p);

//Arquivo perfis.c
int changeInfo(Prof_Node *pn, Profile p) { // altera informações de um perfil
já cadastrado;
    if (pn == NULL) return 0;
    strcpy(pn->info.name, p.name);
    strcpy(pn->info.age, p.age);
    return 1;
}
```

2.2.9 Libera lista de perfis (freeProfiles)

Libera a lista de perfis. É usada apenas quando o programa termina, dentro da função freeList().

a) Verifica se a lista de perfis existe.
b) Declara um ponteiro auxiliar.
c) Percorre a lista de perfis.
d) Guarda o nó atual.
e) Avança para o próximo pefil
f) Libera a lista de séries do perfil e depois libera o perfil.

```
//Arquivo perfis.h
void freeProfiles(Prof_Node* pn);

//Arquivo perfis.c
void freeProfiles(Prof_Node* pn) { // libera lista de perfis
    if (pn != NULL) {
        Prof_Node *aux;
        while (pn != NULL) {
            aux = pn;
            pn = pn->next;
            freeShows(aux->start); // libera a lista de séries do perfil;
            free(aux); // libera perfil
        }
        //free(pn);
    }
}
```

2.2.10 Libera lista (freeList)

Função que libera todas as listas.

a) Verifica se o ponteiro para estrutura não é nulo.
b) Chama a função freeProfiles para liberar a lista começando do *start*.
c) Libera a própria estrutura.

```
//Arquivo perfis.h
void printMainMenu();

//Arquivo perfis.c
void freeList(D_profiles* dp) { // libera todas as listas (usa quando finaliza
o programa);
    if (dp != NULL) {
        freeProfiles(dp->start);
    }
```

```
    free(dp);
}
```

2.2.11 Imprimir menu principal (printMainMenu)

Função que imprime o menu principal.

```
//Arquivo perfis.h
void printMainMenu();

//Arquivo perfis.c
void printMainMenu() { // opções explicadas na main
    printf("\n----- MENU PERFIS -----\n");
    printf("1- Adicionar Perfil\n");
    printf("2- Carregar Perfis\n");
    printf("3- Remover Perfil\n");
    printf("4- Abrir Perfil\n");
    printf("5- Imprimir Perfis\n");
    printf("6- Procurar Perfil\n");
    printf("7- Alterar Dados de um Perfil\n");
    printf("0- Sair\n");
    printf("Selecione uma opcao: ");
}
```

2.2.12 Imprimir menu do perfil (printMenuProfiles)

Função que imprime menu dentro do perfil.

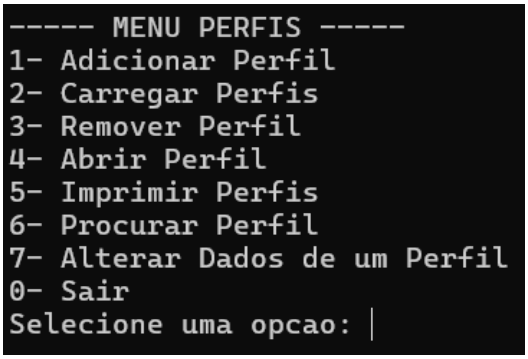
```
//Arquivo perfis.h
void printMenuProfiles(const char *name);

//Arquivo perfis.c
void printMenuProfiles(const char *name) { // opções explicadas na main
    printf("\n----- Perfil de %s -----\n", name);
    printf("1- Adicionar Serie\n");
    printf("2- Carregar Series\n");
    printf("3- Remover Serie\n");
    printf("4- Imprimir Lista de Series\n");
    printf("5- Imprimir Favoritos\n");
    printf("6- Procurar Serie\n");
    printf("7- Adicionar Favorito\n");
    printf("8- Remover Favorito\n");
    printf("0- Sair do Perfil\n");
    printf("Selecione uma opcao: ");
}
```

2.3 Exemplos de Uso

2.3.1 Menu dos perfis

Ao iniciar o programa, será impresso o menu principal (menu perfis) com algumas opções e esperando a seleção de uma opção.

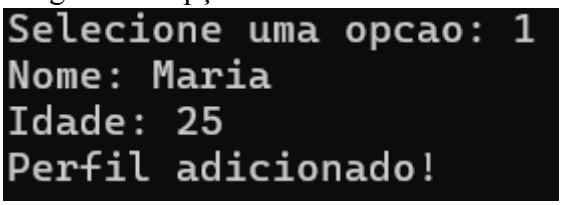
Imagem 4 – Menu dos perfis



Fonte: Autoria própria.

a) Opção 1**:** pergunta os dados de um novo perfil e o adiciona ao fim da lista.

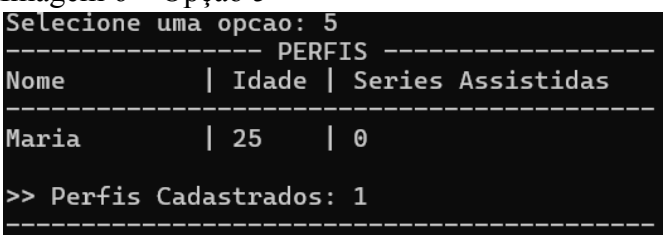
Imagem 5 – Opção 1



Fonte: Autoria própria.

Se você imprimir os perfis agora (Opção 5), você veria algo tipo:

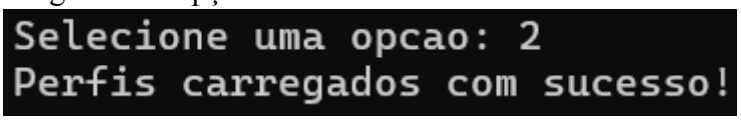
Imagem 6 – Opção 5



Fonte: Autoria própria.

b) Opção 2: carrega um arquivo com perfis modelo (“profiles.txt”) apenas para preencher a lista duplamente encadeada.

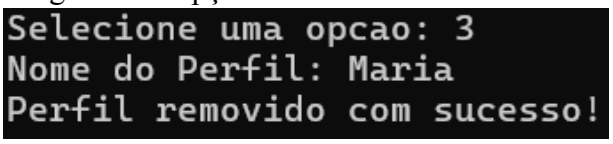
Imagem 7 – Opção 2



Fonte: Autoria própria.

c) Opção 3: procura um perfil a partir do nome e o remove da lista se ele existir. Na imagem, o perfil removido é o que foi adicionado na imagem da opção 1.

Imagem 8 – Opção 3

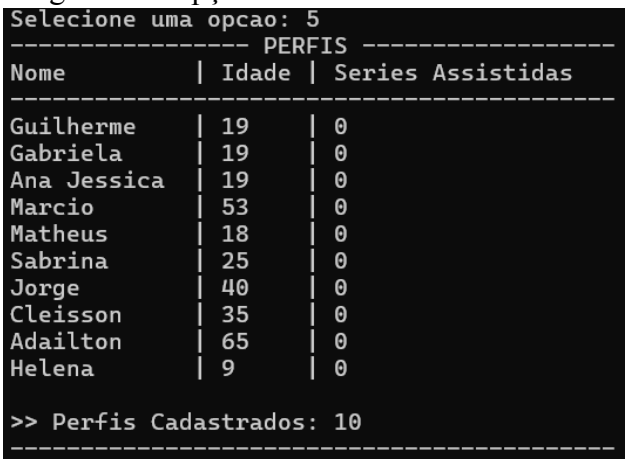


Fonte: Autoria própria.

d) Opção 4: pergunta o nome de um perfil já adicionado na lista e o abre (mais detalhes abaixo).

e) Opção 5: imprime lista com todos os perfis (mostra nomes, idade e quantidade de séries assistidas).

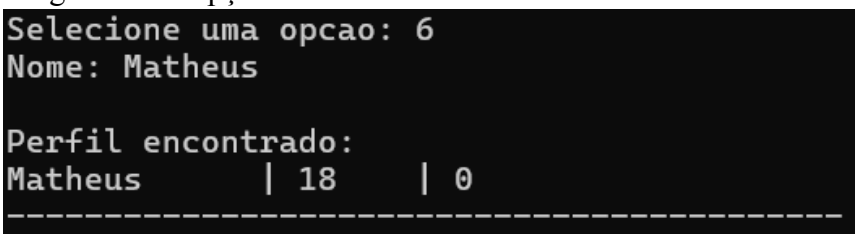
Imagem 9 – Opção 5



Fonte: Autoria própria.

f) Opção 6: procura um perfil a partir de um nome e o exibe se ele existir.

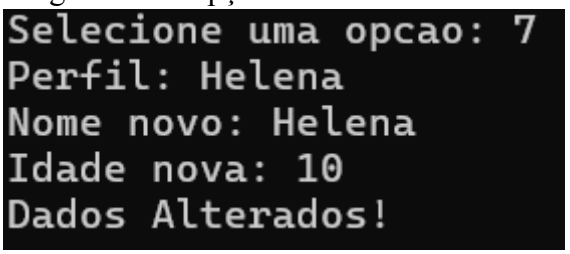
Imagem 10 – Opção 6



Fonte: Autoria própria.

g) Opção 7: procura um perfil a partir de um nome e recebe do usuário um nome e uma idade para alterar suas informações.

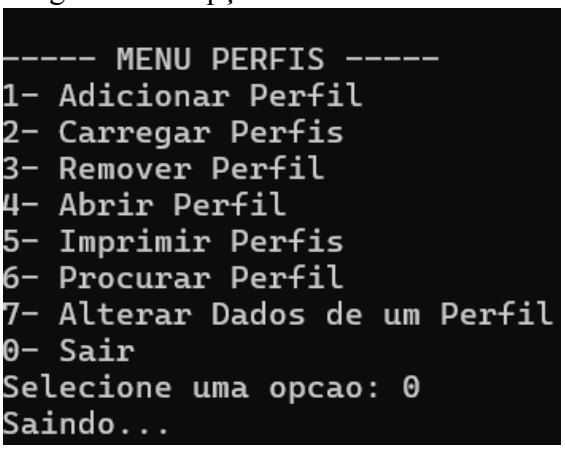
Imagem 10 – Opção 7



Fonte: Autoria própria.

h) Opção 0: libera todas as listas e fecha o programa.

Imagem 11 – Opção 0

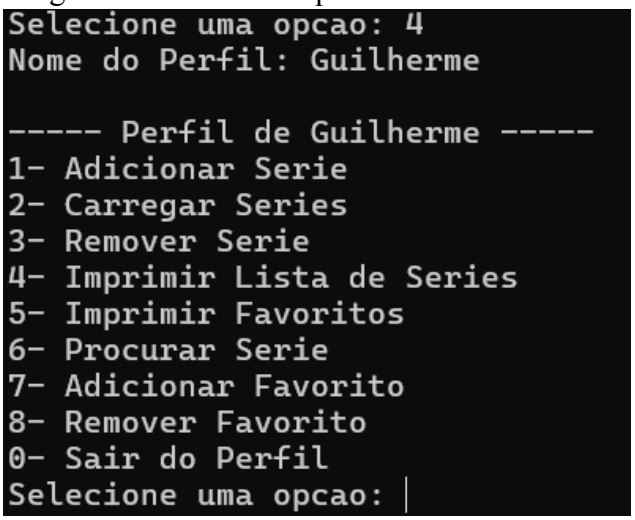


Fonte: Autoria própria.

2.3.2 Menu das séries

Caso escolha a opção (4- Abrir Perfil), você irá inserir um nome de um perfil já cadastrado e você verá algo assim:

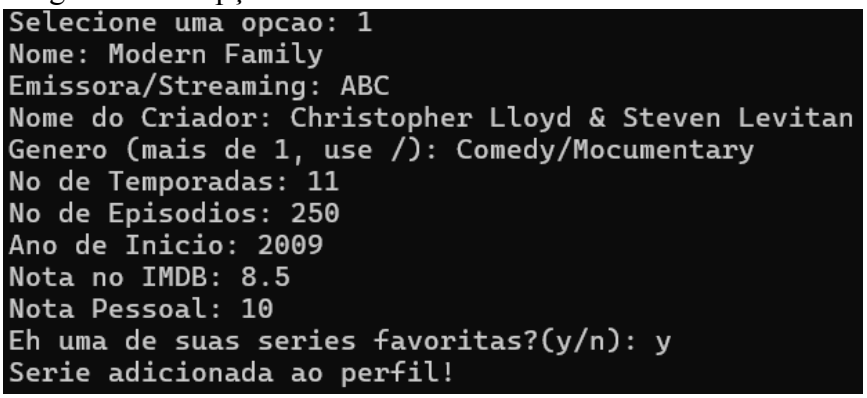
Imagem 12 – Menu do perfil



Fonte: Autoria própria.

a) Opção 1: pergunta os dados de uma série e a adiciona ao fim da lista.

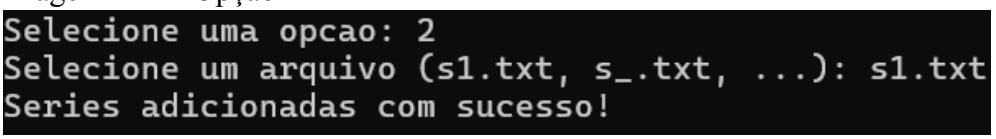
Imagem 13 – Opção 1



Fonte: Autoria própria.

b) Opção 2: carrega um arquivo com uma lista pré-estabelecida* de séries apenas para preencher a lista duplamente encadeada.

Imagem 14 – Opção 2



Fonte: Autoria própria.

Se você imprimir a lista agora (OP4), você veria algo assim:

Imagem 15 – Opção 4

```
Selecione uma opcao: 4

Series de Guilherme:

S2 Modern Family (2009)
Emissora: ABC | Criador(es): Christopher Lloyd & Steven Levitan
No de Temporadas: 11 | No de Episodios: 250
Categoria: Comedy/Mocumentary
Nota IMDB: 8.5 | Nota Pessoal: 10
-----------------------------------

S2 The Wire (2002)
Emissora: HBO | Criador(es): David Simon
No de Temporadas: 5 | No de Episodios: 60
Categoria: Drama/Crime
Nota IMDB: 9.3 | Nota Pessoal: 10
-----------------------------------

Mad Men (2007)
Emissora: AMC | Criador(es): Matthew Weiner
No de Temporadas: 7 | No de Episodios: 92
Categoria: Drama
Nota IMDB: 8.6 | Nota Pessoal: 7
-----------------------------------

S2 Breaking Bad (2008)
Emissora: AMC | Criador(es): Vince Gilligan
No de Temporadas: 5 | No de Episodios: 62
Categoria: Drama/Crime
Nota IMDB: 9.5 | Nota Pessoal: 9
-----------------------------------

Fleabag (2016)
Emissora: BBC Three/BBC One | Criador(es): Phoebe Waller-Bridge
No de Temporadas: 2 | No de Episodios: 12
Categoria: Comedy/Drama
```

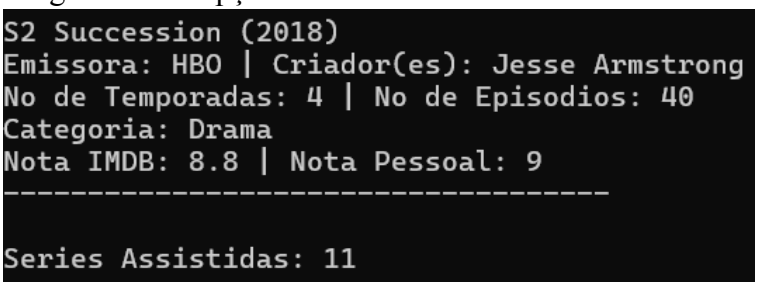
Fonte: Autoria própria.

Imagem 16 – Opção 4

```
S2 Game of Thrones (2011)
Emissora: HBO | Criador(es): David Benioff & D.B. Weiss
No de Temporadas: 8 | No de Episodios: 73
Categoria: Fantasy/Drama
Nota IMDB: 9.2 | Nota Pessoal: 6
-----------------------------------

I May Destroy You (2020)
Emissora: BBC One | Criador(es): Michaela Coel
No de Temporadas: 1 | No de Episodios: 12
Categoria: Drama
Nota IMDB: 8.4 | Nota Pessoal: 7
-----------------------------------

S2 The Leftovers (2014)
Emissora: HBO | Criador(es): Damon Lindelof
No de Temporadas: 3 | No de Episodios: 28
Categoria: Drama/Sci-fi
Nota IMDB: 8.7 | Nota Pessoal: 8
-----------------------------------

The Americans (2013)
Emissora: FX | Criador(es): Joe Weisberg
No de Temporadas: 6 | No de Episodios: 75
Categoria: Espionage/Drama
Nota IMDB: 8.4 | Nota Pessoal: 8
-----------------------------------

S2 The Office (UK) (2001)
Emissora: BBC Two/BBC One | Criador(es): Ricky Gervais & Stephen Merchant
No de Temporadas: 2 | No de Episodios: 12
Categoria: Comedy
Nota IMDB: 8.5 | Nota Pessoal: 7
-----------------------------------
```

Fonte: Autoria própria.

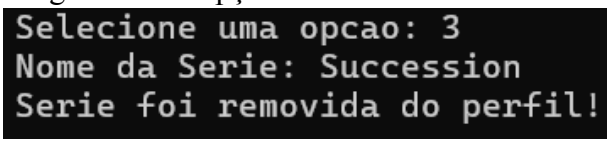
Imagem 17 – Opção 4



Fonte: Autoria própria.

c) Opção 3: procura uma série a partir do nome e a remove da lista de séries do perfil se ela existir.

Imagem 18 – Opção 3

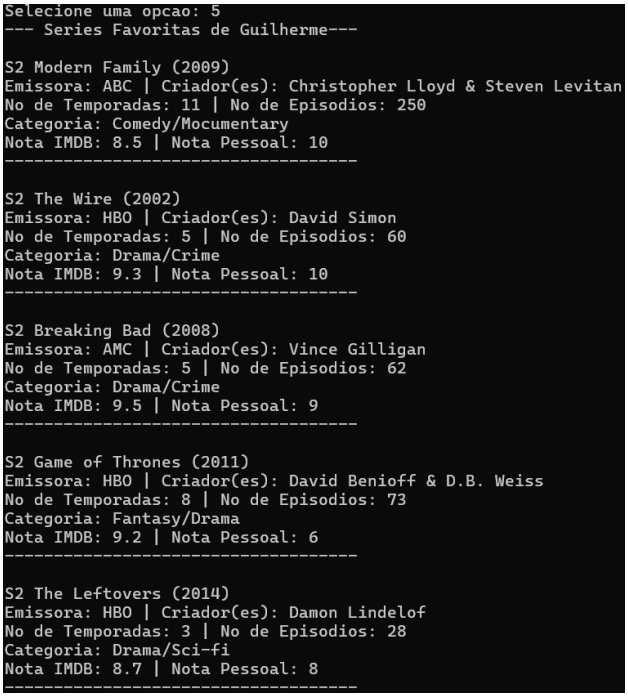


Fonte: Autoria própria.

d) Opção 4: imprime a lista de séries do perfil que o usuário está (Imagem detalhada em OP2).

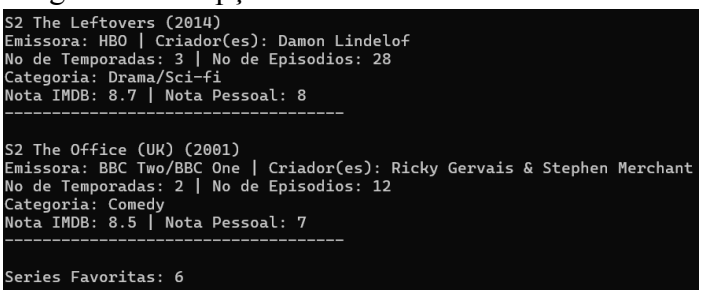
e) Opção 5: imprime a lista de séries favoritas do perfil que o usuário está (Observe que Succession, que foi removida e era favorita, não está nessa lista).

Imagem 19 – Opção 5



Fonte: Autoria própria.

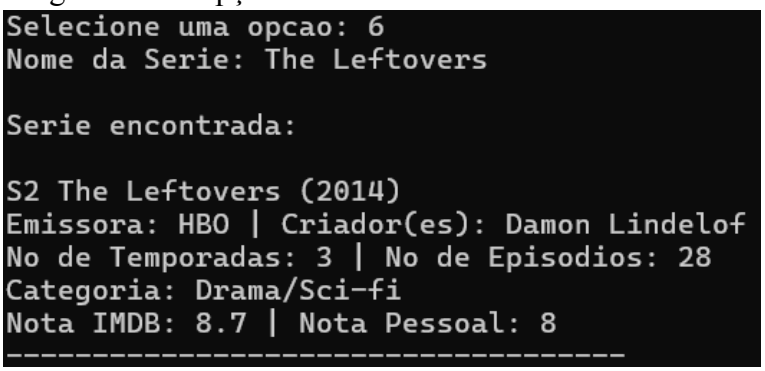
Imagem 20 – Opção 5



Fonte: Autoria própria.

f) Opção 6: procura uma série a partir de um nome e a exibe se ele existir.

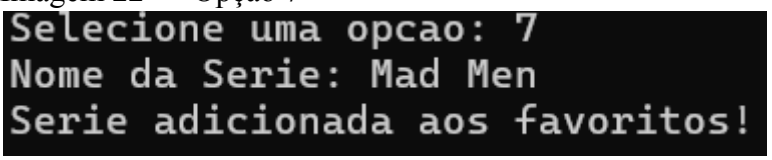
Imagem 21 – Opção 6



Fonte: Autoria própria.

g) Opção 7: recebe o nome da série, procura na lista e a adiciona aos favoritos (se ela já estiver nos favoritos ele retorna que ela já está, e se a série não existir ele retorna que a série não está na lista).

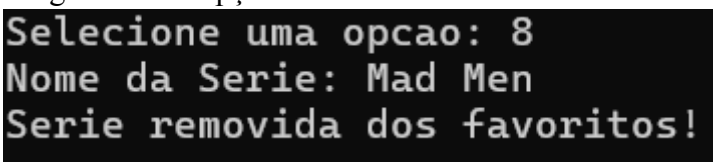
Imagem 22 – Opção 7



Fonte: Autoria própria.

h) Opção 8: recebe o nome da série, procura na lista e a remove dos favoritos (se ela já não estiver nos favoritos ele retorna que ela não é favorita, e se a série não existir ele retorna que a série não está na lista).

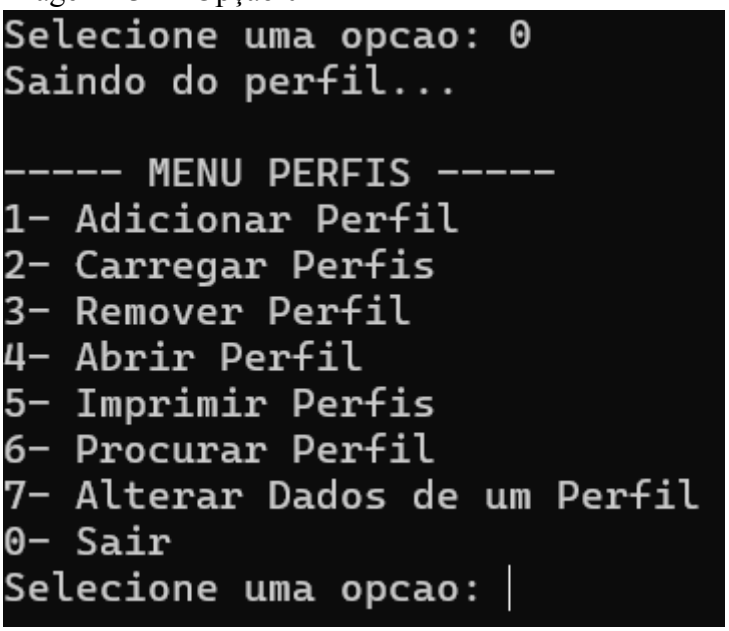
Imagem 23 – Opção 8



Fonte: Autoria própria.

i) Opção 0: Sai do perfil e volta para o Menu Perfis.

Imagem 23 – Opção 0



Fonte: Autoria própria.

3 CONCLUSÃO

Entre os desafios e lições enfrentados durante a implementação do projeto, a reutilização de trechos já desenvolvidos foi essencial para estruturação do código, evitando repetição e promovendo uma organização funcional. Além disso, encontrar soluções simples, quando bem aplicadas, trazem clareza, objetividade e evitam complexidade desnecessária no desenvolvimento. .

REFERÊNCIAS

PROGRAMAÇÃO DESCOMPLICADA. *Programação Descomplicada Linguagem C*. YouTube, @progdescomplicada. Disponível em: https://www.youtube.com/@progdescomplicada. Acesso em: 04 set. 2025.