Trabalho Final - Algoritmos e Estruturas de Dados I

Alunos:

- Leandro Evaristo de Sousa - 12421BCC092
- Mateus Alves Viegas - 12421BCC075

Link GitHub: github.com/leandrosousa007

Introdução

A escolha do tema **Cursos** foi motivada pela nossa familiaridade com o ambiente da faculdade e pela observação de como diferentes cursos frequentemente compartilham disciplinas. O problema central consiste em modelar e gerenciar a relação hierárquica entre cursos e suas respectivas listas de disciplinas de forma eficiente.

A solução foi projetada com foco em **boas práticas** de programação, especialmente a **modularização**. O projeto foi estruturado em um total de **dez arquivos**, permitindo uma clara separação de responsabilidades e facilitando a manutenção do código. Os arquivos são:

- **main.c**: Ponto de entrada do programa, responsável apenas por inicializar as estruturas, chamar o menu principal e liberar a memória ao final.
- **menu.h / menu.c**: Contém a interface do usuário, incluindo a exibição de menus, a captura de entradas e a função de carregamento de dados a partir de um arquivo.
- **lista.h / lista.c**: Implementam a estrutura de dados central do projeto: uma lista duplamente encadeada genérica, capaz de armazenar qualquer tipo de dado.
- **curso.h / curso.c**: Definem a estrutura Curso e implementam todas as funções específicas para sua manipulação.
- **disciplina.h / disciplina.c**: Definem a estrutura Disciplina e implementam as funções para gerenciá-la.
- **teste.txt**: Arquivo de texto contendo dados pré-definidos de cursos e disciplinas, utilizado para inserir no sistema e facilitar a realização de testes.

Dessa forma, foi possível construir um programa de gerenciamento, onde a lógica da estrutura de dados é completamente independente.

Estruturas de Dados (Programa Base)

Para modelar o problema, foram utilizadas estruturas de dados interconectadas. A base de tudo é uma lista duplamente encadeada genérica, que é então utilizada para criar uma lista de cursos, onde cada curso, por sua vez, possui sua própria lista de disciplinas.

A estrutura Lista atua como um **Nó Descritor**, controlando o acesso e as informações gerais de cada lista (início, fim e tamanho).

Estruturas Genéricas (lista.h):

```
typedef struct No {
    void* info;
    struct No* ant;
    struct No* prox;
} No;

typedef struct {
    No* inicio;
    No* fim;
    int tamanho;
} Lista;
```

Estruturas Específicas (curso.h e disciplina.h):

```
typedef struct {
    char nome[100];
    char sigla[10];
    char faculdade[100];
    int vagas;
    int favorito;
    Lista* disciplinas; // Cada curso contém sua própria lista
} Curso;

typedef struct {
    char codigo[20];
    char nome[100];
    char professor[100];
    int carga_horaria;
    int quantidade_alunos;
    int favorita;
} Disciplina;
```

Documentação do Código e Escolha das Funções

Módulo Lista (Genérico)

Estas funções manipulam a lista duplamente encadeada de forma genérica, sem conhecer os dados (Curso ou Disciplina) que estão armazenando.

- **Lista* criar_lista()**
    - **Explicação:** Aloca memória para a estrutura descritora da lista (Lista) e inicializa seus ponteiros inicio e fim como NULL e seu tamanho como 0, deixando-a pronta para uso.
- **void inserir_no_fim(Lista* lista, void* info)**
    - **Explicação:** Adiciona um novo nó ao final da lista. A função aloca um novo No, armazena o ponteiro info nele e ajusta os ponteiros prox do antigo último elemento e ant do novo elemento.
    - **Validação:** Verifica se os ponteiros lista e info não são nulos antes de prosseguir.
- **int remover_no(Lista* lista, const void* criterio, int (*comparar)(const void*, const void*), void (*liberar)(void*))**
    - **Explicação:** Remove o primeiro nó da lista que atende a um critério. Esta função é genérica graças ao uso de ponteiros de função: comparar é uma função externa que sabe como comparar o dado do nó com o critério, e liberar é uma função que sabe como liberar a memória do dado armazenado. A função ajusta os ponteiros dos nós vizinhos para manter a integridade da lista.
    - **Validação:** Retorna 0 (falha) se a lista, o critério ou a função de comparação forem nulos.
- **void destruir_lista(Lista** lista_ref, void (*liberar_info)(void*))**
    - **Explicação:** Libera toda a memória alocada por uma lista. Ela percorre todos os nós, e para cada um, chama a função liberar_info para liberar a memória do dado contido (ex: um Curso), e em seguida libera o próprio nó. Ao final, libera a estrutura da lista e atribui NULL ao ponteiro original para evitar seu uso indevido.

Módulo Curso

Funções que gerenciam exclusivamente os dados dos cursos.

- **Curso* criar_curso(char* nome, char* sigla, char* faculdade, int vagas)**
    - **Explicação:** Aloca memória para uma nova estrutura Curso e a inicializa com os dados fornecidos, além de criar uma lista de disciplinas vazia para o novo curso.
    - **Validação:** Retorna NULL se algum dos ponteiros de texto for nulo ou se o número de vagas for menor ou igual a zero.
- **void liberar_curso(void* info)**
    - **Explicação:** Função de callback projetada para ser usada por destruir_lista ou remover_no. Ela primeiro destrói a lista de disciplinas interna do curso para evitar vazamento de memória e depois libera a memória da própria estrutura Curso.
- **void favorito_curso(Lista* cursos, const char* sigla)**
    - **Explicação:** Percorre a lista de cursos e, ao encontrar o curso com a sigla correspondente, inverte seu campo favorito (de 0 para 1, ou 1 para 0).
- **void listar_curso_com_mais_alunos(Lista* cursos)**
    - **Explicação:** Implementa um algoritmo de duas passadas. Na primeira, percorre todos os cursos para descobrir qual é o número máximo de alunos (somando os alunos de todas as suas disciplinas). Na segunda passada, imprime todos os cursos que possuem esse número máximo de alunos.
- **void mostrar_alunos_curso(Lista* cursos, const char* siglaCurso)**
    - **Explicação:** Busca por um curso específico e, se encontrado, utiliza a função auxiliar calcular_total_alunos para somar os estudantes de todas as suas disciplinas e exibir o total.

Módulo Disciplina

Funções que gerenciam exclusivamente os dados das disciplinas.

- **Disciplina* criar_disciplina(char* codigo, char* nome, char* professor, int carga, int alunos)**

    - **Explicação:** Aloca memória para uma nova Disciplina e preenche seus campos com os dados recebidos.
    - **Validação:** Retorna NULL se algum ponteiro de texto for nulo, se a carga horária for negativa ou se a quantidade de alunos for negativa.
- **void listar_disciplinas_mais_inscritas(Lista* cursos)**
    - **Explicação:** Similar à função de curso com mais alunos, esta também utiliza um algoritmo de duas passadas para primeiro encontrar a maior quantidade de alunos em uma única disciplina e depois listar todas as disciplinas que atingem esse valor.

Módulo Menu

Funções responsáveis pela interação com o usuário e manipulação de arquivos.

- **void carregar_dados_de_arquivo(Lista* cursos)**
    - **Explicação:** Abre o arquivo teste.txt em modo de leitura. Lê o arquivo linha por linha, utilizando a função strtok para separar os dados pelo delimitador ";". Com base no primeiro caractere ('C' ou 'D'), cria e insere um novo curso ou adiciona uma nova disciplina ao último curso criado.
    - **Validação:** Verifica se o arquivo teste.txt existe; caso contrário, informa o usuário e continua a execução com o sistema vazio.
- **void menu_principal(Lista* cursos)**
    - **Explicação:** Funciona como o centro de navegação principal do programa. Exibe um laço com as opções para gerenciar cursos, gerenciar disciplinas, ver relatórios ou sair do sistema. De acordo com a escolha do usuário, ela direciona a execução para o submenu apropriado (menu_cursos, menu_disciplinas, etc.).
    - **Validação:** Possui um caso default em sua estrutura switch para tratar entradas numéricas que não correspondem a uma opção válida, informando o usuário sobre a escolha inválida.
- **void menu_cursos(Lista* cursos)**
    - **Explicação:** Apresenta um submenu dedicado às operações de cursos: inserir, listar, favoritar e remover. Para cada ação, a função solicita ao usuário os dados necessários (como nome e sigla), invoca

as funções de lógica de negócio correspondentes (ex: criar_curso, remover_no) e exibe uma mensagem de sucesso ou erro.
    - **Validação:** A validação dos dados de entrada (ex: vagas > 0) é delegada às funções de criação (criar_curso), enquanto a busca por itens para remover ou favoritar é tratada pelas respectivas funções de lógica.
- **void menu_disciplinas(Lista* cursos)**
    - **Explicação:** Gerencia todas as ações relacionadas a disciplinas. Para operações como inserção ou remoção, a função primeiro solicita a sigla do curso ao qual a disciplina pertence. Após localizar o curso, ela executa a operação desejada (adicionar, listar, favoritar ou remover) na lista de disciplinas interna daquele curso específico.
    - **Validação:** Realiza uma verificação crucial para garantir que o curso especificado pelo usuário realmente existe antes de prosseguir com qualquer operação de disciplina. Se o curso não for encontrado, uma mensagem de erro é exibida.
- **void menu_relatorios(Lista* cursos)**
    - **Explicação:** Centraliza a geração de todos os relatórios do sistema. Oferece ao usuário um menu de opções onde cada uma aciona uma função específica para extrair e exibir informações, como a listagem completa de todos os dados, a exibição apenas de itens favoritados ou a identificação de cursos e disciplinas com maior número de alunos.
    - **Validação:** Assim como o menu principal, contém um caso default para capturar e informar o usuário sobre escolhas de menu inválidas.

Funções de Comparação

- **int comparar_curso_por_sigla(const void* curso_info, const void* sigla)**
    - **Explicação:** Atua como uma função de callback que recebe um ponteiro genérico para um curso e um para uma sigla. Ela converte os ponteiros para seus tipos corretos (Curso* e char*) e utiliza strcmp para verificar se a sigla do curso corresponde à sigla fornecida. Sua finalidade é permitir que a função remover_no encontre um curso específico para remoção.

    - **Validação:** A lógica principal desta função é a própria comparação. Ela retorna 0 se os identificadores forem iguais, sinalizando uma correspondência para a função que a chamou.
- **int comparar_disciplina_por_codigo(const void* disciplina_info, const void* codigo)**
    - **Explicação:** Semelhante à anterior, esta função de callback ensina remover_no a identificar uma disciplina. Ela converte os ponteiros genéricos para Disciplina* e char* e compara o código da disciplina com o critério fornecido, usando strcmp.
    - **Validação:** Seu retorno (0 para correspondência) é o que valida se o nó correto foi encontrado para a remoção.

Exemplos de Uso

Lista os curso favoritos

```
=== RELATÓRIOS ===
1. Listar todos os cursos e disciplinas
2. Cursos favoritos
3. Disciplinas favoritas
4. Disciplinas mais inscritas
5. Curso com mais alunos
6. Quantidade de alunos na disciplina
7. Quantidade de alunos no curso
0. Voltar
Escolha: 2

=== CURSOS FAVORITOS ===

----------------------------------------
Ciencia da Computacao (CIC) - FACOM, Vagas: 50 (FAVORITO)
 Disciplinas:
[GBC021] Algoritmos e Estruturas de Dados I - Professor: Joao Santos, Carga: 90, Alunos: 45
[GBC033] Organizacao de Computadores - Professor: Ana Pereira, Carga: 60, Alunos: 40
[GSI003] Matematica para Ciencia da Computacao - Professor: Maria Silva, Carga: 60, Alunos: 55
[GBC045] Redes de Computadores - Professor: Lucas Ferreira, Carga: 60, Alunos: 38
----------------------------------------

----------------------------------------
Sistemas de Informacao (SI) - FACOM, Vagas: 50 (FAVORITO)
 Disciplinas:
[GSI003] Matematica para Ciencia da Computacao - Professor: Maria Silva, Carga: 60, Alunos: 55
[GSI023] Engenharia de Requisitos - Professor: Carlos Lima, Carga: 60, Alunos: 50 (FAVORITA)
[GSI025] Banco de Dados - Professor: Beatriz Costa, Carga: 60, Alunos: 48
----------------------------------------
```

Imprime a disciplina com o maior número de alunos inscritos

```
=== RELATÓRIOS ===
1. Listar todos os cursos e disciplinas
2. Cursos favoritos
3. Disciplinas favoritas
4. Disciplinas mais inscritas
5. Curso com mais alunos
6. Quantidade de alunos na disciplina
7. Quantidade de alunos no curso
0. Voltar
Escolha: 4

=== DISCIPLINAS MAIS INSCRITAS ===
Disciplinas com mais inscrições (78 alunos):
Curso: MED | [GME005] Fisiologia Humana - Professor: Marcelo Tavares, Carga: 90, Alunos: 78
```

Lista as disciplinas favoritas

```
=== RELATÓRIOS ===
1. Listar todos os cursos e disciplinas
2. Cursos favoritos
3. Disciplinas favoritas
4. Disciplinas mais inscritas
5. Curso com mais alunos
6. Quantidade de alunos na disciplina
7. Quantidade de alunos no curso
0. Voltar
Escolha: 3

=== DISCIPLINAS FAVORITAS ===
[GSI023] Engenharia de Requisitos - Professor: Carlos Lima, Carga: 60, Alunos: 50 (FAVORITA)
```

Imprime a quantidade total de alunos no curso de Ciência da Computação

```
=== RELATÓRIOS ===
1. Listar todos os cursos e disciplinas
2. Cursos favoritos
3. Disciplinas favoritas
4. Disciplinas mais inscritas
5. Curso com mais alunos
6. Quantidade de alunos na disciplina
7. Quantidade de alunos no curso
0. Voltar
Escolha: 7
Sigla do curso: CIC
O curso 'Ciencia da Computacao' tem 178 aluno(s) matriculado(s) no total.
```

Insere o Curso de Inteligência Artificial

```
=== GERENCIAR CURSOS ===
1. Inserir Curso
2. Listar Cursos
3. Favoritar Curso
4. Remover Curso
0. Voltar
Escolha: 1
Nome do curso: Inteligencia Artificial
Sigla do curso: IA
Faculdade: UFU
Vagas: 60
Curso 'IA' inserido com sucesso!
```

Remove o curso de Inteligência Artificial

```
=== GERENCIAR CURSOS ===
1. Inserir Curso
2. Listar Cursos
3. Favoritar Curso
4. Remover Curso
0. Voltar
Escolha: 4
Digite a SIGLA do curso a ser removido: IA
Curso 'IA' removido com sucesso!
```

Conclusão

Durante o desenvolvimento, enfrentamos alguns desafios que foram fundamentais para o nosso aprendizado, como a repetição de funções para inserir, criar, etc. Para evitar essa subutilização desnecessária, criamos um nó genérico. Embora tenha sido difícil conceber seu funcionamento no início, a implementação foi bem-sucedida.

A função de remoção também foi trabalhosa, pois, ao criá-la de forma genérica, precisamos desenvolver outras duas funções auxiliares para comparar a informação fornecida (se era a sigla ou o código). Integrar tudo isso demandou um tempo considerável.

Por fim, este projeto reforçou a importância da modularização e de um bom planejamento inicial. A decisão de dividir o código em múltiplos arquivos desde o começo tornou o desenvolvimento mais organizado e facilitou o trabalho em equipe. As lições aprendidas sobre alocação dinâmica, ponteiros и design de software serão, sem dúvida, valiosas para projetos futuros. Além disso, a utilização do GitHub para armazenar a documentação e os códigos facilitará nosso trabalho na plataforma futuramente.