UNIVERSIDADE FEDERAL DE UBERLÂNDIA

ANA LUÍZA MOREIRA SILVA

JEAN LUC GIRVENT DEU

MATHEUS MARTINS DE RESENDE

CONTROLE DE TRENS E VAGÕES

UBERLÂNDIA

2025

1 INTRODUÇÃO

O trabalho foi elaborado com o intuito de desenvolver uma aplicação de gerenciamento de uma ferrovia, tendo acesso aos trens e seus respectivos vagões. O objetivo é permitir ao usuário adicionar e remover trens, através de um número identificador (ID) único, além de gerenciar os vagões presentes em cada comboio, também por meio de um ID, controlando a adição e remoção em qualquer posição desejada, atribuindo a cada vagão informações sobre o tipo de carga presente e sua capacidade máxima.

Para criar a aplicação foi necessário organizar a adição dos trens (cada trem é armazenado como uma estrutura), por meio de uma lista simplesmente encadeada, que conforme estudado na disciplina de Algoritmos e Estruturas de Dados 1 (AED 1), os elementos da lista se conectam por meio de uma variável do tipo ponteiro que armazena o endereço da próxima estrutura que será adicionada. O processo para adição dos vagões foi feito de maneira semelhante, porém com o uso de lista duplamente encadeada, que além de armazenar um ponteiro para a próximo vagão, também armazena um ponteiro para o vagão anterior.

Primeiramente foi necessária a criação de uma lista vazia para adição dos trens e uma para adição dos vagões. A inserção de um novo trem foi feita ao início da lista, sempre atualizando o ponteiro que armazena o endereço do próximo trem e para um novo vagão o usuário poderia escolher onde desejava adicioná-lo, sendo necessário percorrer a lista de vagões pré-existentes até encontrar a posição solicitada e atualizar os ponteiros que armazenam o endereço do vagão anterior e do seguinte tanto para a estrutura que foi inserida, como da anterior a ela e a posterior.

Em seguida, para a remoção de um trem ou de um vagão o processo foi semelhante. Por meio do identificador foi realizada a busca do elemento desejado, reajustando os ponteiros das estruturas anteriores e posteriores, no caso da lista simplesmente encadeada foi ajustado os ponteiros para o endereço do próximo e para a lista duplamente encadeada foram ajustados os ponteiros para o endereço do próximo e do anterior. Por fim foi necessário liberar a memória referente a estrutura que foi removida.

Outra coisa, a modularização do código foi feita de forma onde as funções de manipulação de trem ficam separadas do arquivo onde fica a função Main, logo, para a chamada das funções tem um arquivo intermediário que faz a ponte entre a as funções, mantendo uma fácil manutenção, compreensão e reutilizabilidade do código.

Para acesso ao repositório do código, acesse: **https://github.com/girvent27/Trab-AED1**

2 DOCUMENTAÇÃO DO CÓDIGO

2.1 DOCUMENTAÇÃO DA MAIN

Criação de variáveis na main.c e chamada de todas as outras funções de func.c e node.c a partir de um laço de repetição do while para criação de um menu interativo com o usuário. Dentro da função Do existe um switch com os casos 1 a 8 que permite ao usuário escolher entre funções do nosso programa, o que será abordado melhor em **3. EXEMPLOS DE USO.**

2.2 DOCUMENTAÇÃO DO HEAD

Head possui as assinaturas das funções em node.h e func.h bem como os seguintes tipos abstratos de dados:

1- Struct sNode

- Possui os campos de informação: id, vagões, unidade ou quilograma
- Possui o campo struct sNode *prox para implementar um nó simples

2- Struct TrainCar

Possui campos informacionais de um vagão como:

- Carga
- Quantidade da carga
- Unidade de medida.

3- Struct dNode

- Possui o campo vagao de informações e id
- Possui os campos struct dNode *next e dNode *prev para implementar o nó duplo.

2.3 DOCUMENTAÇÃO DAS FUNÇÕES

Funções de manipulação do trem e dos vagões presentes em node.c, demais funções para o menu presentes em func.c.

#include "func.h"

- Assinatura das funções presente na HEAD

int maiorID(sNode **trens);

- Declaração da função local para encontrar o maior ID, retornando ela.

sNode* criaTrem(sNode *trem)

- Criação da lista vazia de trens.int adicionaTrem(sNode **trens)

- Função que adiciona um trem ao início de uma lista simplesmente encadeada. Primeiramente é alocada memória para receber os dados da estrutura. Se o trem for o primeiro elemento a ser adicionado, o ponteiro que recebe o endereço do próximo elemento passa a ser NULL, pois ainda não há um próximo elemento. Se não, o ponteiro recebe o endereço do primeiro elemento e o novo trem passará a ser o primeiro elemento da lista. Retorna 1 se sucesso e 0 em caso de erro.

int maiorID(sNode **trens)

- Função que calcula qual o trem com a maior ID, comparando entre todos os presentes na lista. E retorna o maior valor de ID encontrado. Esse dado será utilizado para gerar uma ID para o novo trem adicionado, somando 1 ao maior valor encontrado anteriormente.

void listarTrens(sNode *trens)

- Função que percorre a lista de trens e imprime todos os trens presentes por meio de sua ID. É utilizada uma estrutura auxiliar para percorrer todos os elementos da lista.

int removerTrem(sNode **trens, int id)

- Função para remover um trem da lista. Para isso, é necessário encontrar o trem na lista com o mesmo ID solicitado pelo usuário. Ao percorrer a lista, além de uma variável auxiliar, é utilizado também uma variável que armazena o elemento anterior da lista, assim quando o trem desejado é encontrado os ponteiros são atualizados, o do elemento anterior recebe o endereço do elemento seguinte ao desejado e a memória armazenada para o trem removido é liberada. Se o trem removido for o primeiro basta apenas fazer com que o início da lista passe a ser o elemento seguinte. Retorna 1 se sucesso e 0 em caso de erro.

Int trensVazio(sNode *trens)

- Verifica se a lista está vazia retornando 0 (vazia) ou 1 (cheia).

dNode *criaLocomotiva()

- Essa função foi um complemento para solucionar o problema de um trem necessariamente precisar de uma locomotiva em uma das pontas, logo ela cria automaticamente essa locomotiva e auxilia a função adicionaTrem.

Int listaVagao(sNode **, int, Vagao, int)

- Função usada para imprimir informações de um Trem, ela recebe a ID de um dos trens, utiliza a função buscaTrem, imprime as informações de todos os vagões presentes nesse trem e retorna 1 se bem concluída. Se não houver vagões ou ocorrer erros, retorna 0.

Int insereVagao(sNode *, int, Vagao, int)

- Utiliza de outras 2 funções como maiorIdVagao (que busca o maior id de vagões presente na lista) e buscaTrem (retorna um nó da lista simples de Trens, esse nó representa um Trem com seus vagões e locomotiva) para adicionar um vagão ao trem de ID recebido pela função, caso não exista ela retorna 0.

Usa-se alocação dinâmica de memória para criar o nó do vagão e adicionar a lista duplamente encadeada representada por um Trem. Retorna 1 se sucesso e 0 em caso de erro.

Int mudaCarga (sNode **Trem, int ID, int IdVag)

- Altera os campos Carga e Unidade de um vagão, também utiliza buscaTrem, para modificar o vagão presente dentro de um nó de um dos Trens. Retorna 1 se sucesso e 0 em caso de erro.

Int excluiVagao (sNode **Trem, int ID, int IdVag)

- Deleta um dos nós presentes em uma lista duplamente encadeada que representa um Trem, basicamente busca na lista simplesmente encadeada um Trem que é uma lista duplamente encadeada e nela exclui o vagão de IDVag transferido a função. Retorna 1 se sucesso e 0 em caso de erro.

Int organizarVagao (sNode **Trem, int ID, int IdVag, int posi_nova)

- Função responsável por reorganizar vagões, ela troca um vagão de posição na lista duplamente encadeada mantesndo as mesmas informações do *struct Vagao* e o mesmo ID, isso só ocorre pois há a contagem das posições da lista. Retorna 1 se sucesso e 0 em caso de erro.

Int menu ()

- Responsável pela impressão do Menu principal e apenas isso. Retorna a opção escolhida do menu (de 0 a 9, sendo 9 de teste).

Vagao structVagao (char *, double, char *)

- Função auxiliar que cria um tipo abstrato vagão e anexa informações nos campos de vagão. Retorna uma variável do tipo Vagao.

void teste(sNode **);

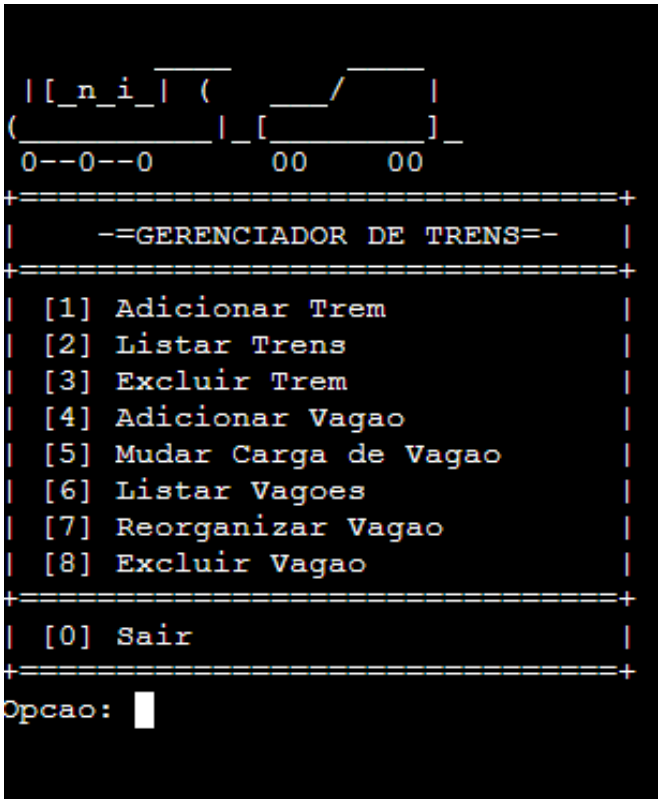
- Cria 4 listas de trens e atribuem a elas uma quantidade sortida de vagões para fins de teste e verificação de funcionabilidade.

As seguintes funções servem de ponte entre a main.c e a nodes.c, para manter a facilidade de leitura e manutenção do código, elas fazem a validação das informações passadas pelos usuários e logo depois chama a respectiva função em node.c passando somente a lista de trens como parâmetro:

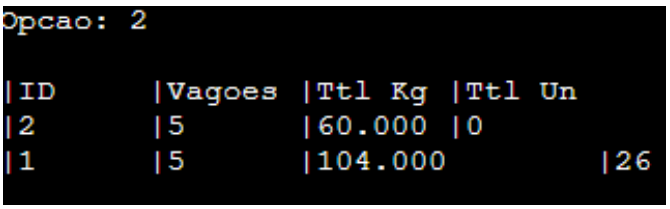
- void mainCriaVagao(sNode *);
- void mainReorganizaVagao(sNode *);
- void mainExcluiVagao(sNode *);
- void mainMudarCarga(sNode *);
- void mainExcluirVagao(sNode *);

3 EXEMPLOS DE USO

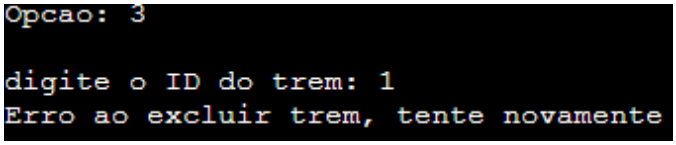
Ao rodar o programa pela primeira vez o usuário é introduzido ao MENU.



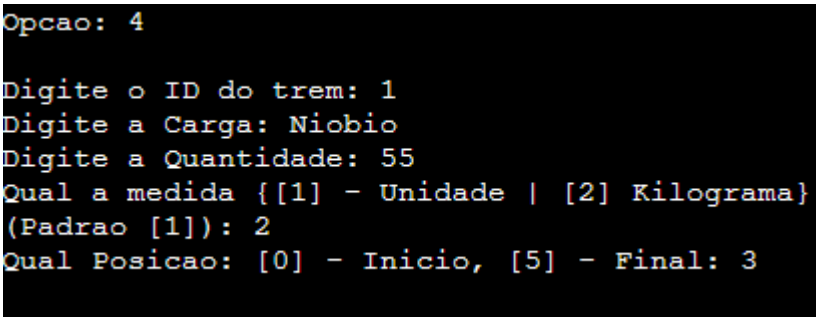
A opção 1 adiciona um Trem a lista simples, é a operação básica realizada dentro do programa, já a opção 2 Lista os Trens, no caso de não ter trens na lista, o programa retorna a mensagem: não há trens criados e retorna ao MENU.



A opção 3 exclui um trem da lista de acordo com o ID, caso seja digitado um ID existente, o programa retorna a mensagem de confirmação da exclusão, porém se o ID não existe na lista, o programa retorna a mensagem: Erro ao excluir trem, tente novamente. E retorna ao MENU.



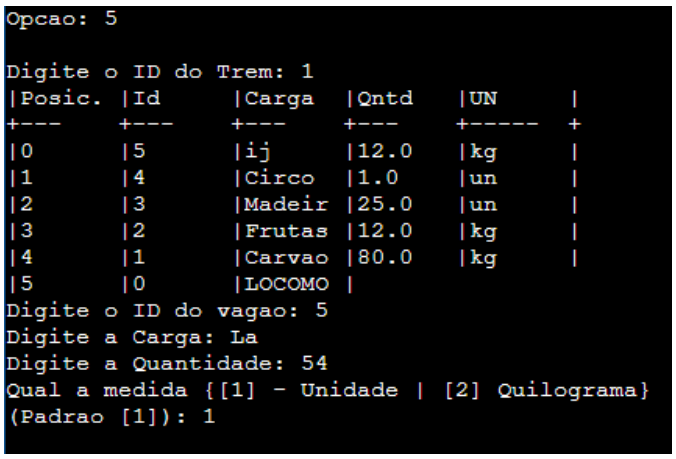
Opção 4 leva o usuário a informar ID do trem a ser adicionado o vagão, o tipo de carga, a quantidade e a unidade de medida, podendo ser (1) unitário ou (2) quilograma, bem como a posição do vagão no trem.



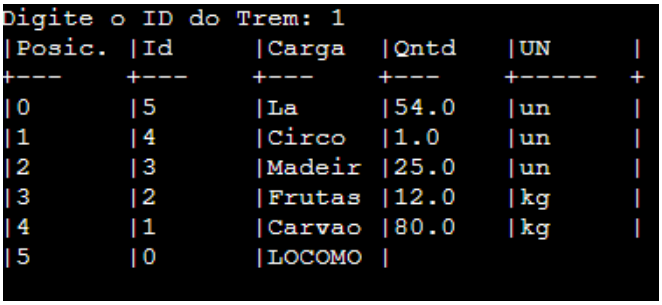
A opção 5 permite alterar o conteúdo dentro de um vagão já existente, o usuário deve informar o ID do trem, o ID do vagão a ser alterado e as informações da carga, como nome, quantidade e unidade de medida. Em conjunto temos a opção 6 que permite o usuário ver os elementos de cada trem a

partir do ID a ser informado, aqui utilizado para ver o resultado após a operação de troca da carga do vagão de ID 5.

Antes da alteração:



Depois da alteração (utilização da opção 6):



A opção 7 é utilizada quando se deseja mudar a ordem dos vagões de um trem, aqui informamos o ID do trem que queremos alterar, bem como o ID do vagão a ser alterado e a posição a ser colocado. Nota-se que se abre um submenu para auxiliar na escolha.

Alteração da posição do vagão de ID 2 e posição 3 para a posição 1:

```
Opcao: 7

Digite o ID do Trem: 1
|Posic.  |Id     |Carga  |Qntd   |UN     |
+---     +---    +---    +---    +-----  +
|0       |5      |ij     |12.0   |kg     |
|1       |4      |Circo  |1.0    |un     |
|2       |3      |Madeir |25.0   |un     |
|3       |2      |Frutas |12.0   |kg     |
|4       |1      |Carvao |80.0   |kg     |
|5       |0      |LOCOMO |
Digite o ID do vagao: 2
Qual Posicao: [0] - Inicio, [4] - Final: 1
```

Após a alteração:

```
|Posic.  |Id     |Carga  |Qntd   |UN     |
+---     +---    +---    +---    +-----  +
|0       |5      |ij     |12.0   |kg     |
|1       |2      |Frutas |12.0   |kg     |
|2       |4      |Circo  |1.0    |un     |
|3       |3      |Madeir |25.0   |un     |
|4       |1      |Carvao |80.0   |kg     |
|5       |0      |LOCOMO |
```

A opção 8 exclui um vagão conforme o ID do trem e ID do vagão a ser eliminado, caso haja algum erro nesta operação aparecerá a mensagem de erro ou de trem inexistente.

```
Opcao: 8

Digite o ID do Trem: 1
|Posic.  |Id     |Carga  |Qntd   |UN     |
+---     +---    +---    +---    +-----  +
|0       |5      |ij     |12.0   |kg     |
|1       |4      |Circo  |1.0    |un     |
|2       |3      |Madeir |25.0   |un     |
|3       |2      |Frutas |12.0   |kg     |
|4       |1      |Carvao |80.0   |kg     |
|5       |0      |LOCOMO |
Digite o ID do vagao: 5
```

4 CONCLUSÃO

Conclui-se que o para o desenvolvimento do trabalho foi de extrema importância elaborar detalhadamente como seria feito o gerenciamento dos vagões e dos trens, pois iniciar a escrita dos códigos antes dessa etapa seria um processo muito mais demorado e com maior possibilidade de eventuais erros. Analisando, primeiramente, com cuidado como as listas seriam organizadas facilitou significativamente o processo, pois com o toda a estruturação criada, foi necessário apenas transformar essa lógica em código.

Uma das maiores dificuldades encontradas ao criar a aplicação foi como designar tarefas a todos os membros de forma justa, sem gerar possíveis déficits de aprendizagem a nenhum dos envolvidos e possibilitando uma colaboração agregativa. Para não prejudicar na fluidez do código alguns padrões de escrita foram adotados. Todo o processo de desenvolvimento foi acompanhado por todos os membros, dessa forma, evitando que houvesse diferença no entendimento entre os envolvidos.