Relatório - Trabalho AED

Grupo:

- **Camilo Saud Gasparino de souza - 12411BCC023**
- **Pedro Antonio Carrijo Batista - 12411BCC007**
- **Iury Spirlandelli Hattori - 12411BCC040**

Introdução

O objetivo deste trabalho foi desenvolver um sistema em linguagem C capaz de controlar uma lista de trens, onde cada trem pode conter múltiplos vagões associados. A ideia principal é utilizar listas encadeadas para modelar os trens e listas duplamente encadeadas para os vagões, garantindo flexibilidade na adição, remoção e manipulação dos elementos.

O sistema foi construído utilizando:

1. Uma lista encadeada simples para representar os trens.
2. E para cada trem, uma lista duplamente encadeada

para seus vagões

O sistema possui as seguintes funcionalidades:

1. Inserção ordenada de trens e vagões
2. Adição de vagões no início ou no final da lista.
3. Remoção de trens e vagões.
4. Troca de vagões entre dois trens.
5. Impressão completa da estrutura.

Documentação

Estruturas:

```
typedef struct node{

    int maxQtd;
    char tipoCarga[100];
    struct node *prox;
    struct node *ant;



} NoVagoes;

typedef struct nose {
    char train[100];
    NoVagoes *vagoes;
    struct nose *prox;
} NoTrens;
```

Funções Principais:

1) crialistaTrem: Inicializa a lista de trens com valor NULL.
Propósito: Define uma lista vazia de trens, essencial como ponto de partida para
qualquer operação posterior de inserção.
Construção:

- A função recebe um ponteiro duplo para o início da lista de trens.
- Simplesmente define esse ponteiro como NULL, indicando uma lista vazia.
- Isso é necessário porque toda lista encadeada precisa de um ponto inicial bem definido.

2) criaTrem: Aloca memória para um novo trem, copia o nome fornecido, e inicializa os campos prox e vagoes como NULL.
Propósito: Cria um novo nó de trem para ser inserido na lista principal.
Construção:

- ➢ Usa malloc para alocar espaço para um novo nó NoTrens.
- ➢ A função strcpy copia o nome do trem para o campo train.
- ➢ Inicializa os campos prox (próximo trem) e vagoes (início da lista de vagões) como NULL.
- ➢ Retorna o novo trem pronto para ser inserido.

3) adicionaTrem: Insere um novo trem no início da lista de trens.

Propósito: Permite adicionar rapidamente novos trens no início da estrutura, sem ordenação.

Construção:

- ➢ Chama criaTrem para gerar o novo trem.
- ➢ Ajusta o ponteiro prox para que o novo trem aponte para o antigo início da lista.
- ➢ Atualiza *head para que o novo trem seja o primeiro da lista.
- ➢ Essa abordagem permite inserção em tempo constante

4)adicionarTremOrdenado: Insere um novo trem de forma ordenada alfabeticamente.

Propósito: Mantém a lista de trens organizada por ordem alfabética de nome, facilitando buscas e leitura.

Construção:

- ➢ Cria o novo trem como nas funções anteriores.
- ➢ Se a lista estiver vazia ou o novo nome for alfabeticamente anterior ao primeiro, insere no início.
- ➢ Caso contrário, percorre a lista com um ponteiro auxiliar até encontrar o ponto de inserção baseado na ordem alfabética (usando strcmp).
- ➢ Atualiza os ponteiros para inserir o novo trem no lugar correto.

5) criaVagao: Cria um novo vagão com tipo de carga e quantidade máxima, inicializando ponteiros de conexão (prox, ant) como NULL.

Propósito: Permite a criação de um vagão independente, pronto para ser conectado a um trem.

Construção:

- ➢ Usa malloc para alocar o espaço necessário.

- ➢ Copia o tipo de carga e define a quantidade máxima.
- ➢ Inicializa os ponteiros prox e ant como NULL, pois o vagão ainda não está conectado.
- ➢ Retorna o novo vagão.

6) adicionaVagaoInicio: Adiciona um novo vagão no início da lista de vagões de um trem específico.

Propósito: Facilita inserções rápidas no começo da lista de vagões.

Construção:

- ➢ Localiza o trem desejado pelo nome.
- ➢ Cria um novo vagão com criaVagao.
- ➢ Insere o vagão no início da lista de vagões do trem:
  - ○ O novo vagão aponta para o atual primeiro.
  - ○ Se houver um vagão já existente, seu ponteiro ant é ajustado.
  - ○ O ponteiro vagoes do trem passa a apontar para o novo vagão.

7) adicionarVagaoFinal: Adiciona um novo vagão ao final da lista de um trem.

Propósito: Garante que novos vagões sejam adicionados de forma encadeada ao fim da lista.

Construção:

- Localiza o trem.
- Cria o novo vagão.
- Se a lista de vagões estiver vazia, insere o vagão diretamente.
- Caso contrário, percorre até o último vagão e insere o novo após ele:
  - Atualiza prox do último para apontar para o novo.
  - Define ant do novo para apontar para o antigo último.

8) adicionarVagaoOrdenado: Insere um vagão de forma ordenada com base no tipo de carga.

Propósito: Mantém a lista de vagões organizada, melhorando legibilidade e manutenção.

Construção:

- Localiza o trem desejado.

- ➢ Cria o novo vagão.
- ➢ Se a lista estiver vazia ou o novo tipo de carga for anterior ao primeiro, insere no início.
- ➢ Caso contrário, percorre os vagões até encontrar a posição correta com base no tipo de carga.
- ➢ Ajusta os ponteiros prox e ant para inserir o vagão de forma ordenada..

9) removerTrem: Remove um trem da lista, liberando todos os seus vagões.

Propósito: Evita vazamentos de memória e garante remoção segura da estrutura..

Construção:

- ➢ Percorre a lista de trens para encontrar o que deve ser removido.
- ➢ Mantém um ponteiro anterior para reencadear a lista depois da remoção.
- ➢ Libera todos os vagões ligados ao trem usando um laço.
- ➢ Libera o próprio trem e ajusta os ponteiros para manter a lista funcional.

10) removerVagao: Remove um vagão específico de um trem, mantendo a integridade da lista duplamente encadeada

Propósito: Permite a remoção pontual de cargas, sem afetar os demais vagões

Construção:

- ➢ Localiza o trem e depois o vagão a ser removido.
- ➢ Ajusta os ponteiros prox e ant dos vizinhos para que a lista permaneça válida.
- ➢ Se o vagão for o primeiro, atualiza o ponteiro vagoes do trem.
- ➢ Finalmente, libera o vagão com free.

11) TrocarVagoesEntreTrens: Troca os ponteiros de lista de vagões entre dois trens.

Propósito: Permite reorganizar a estrutura da rede ferroviária de forma flexível.

Construção:

- ➢ localiza os dois trens envolvidos na troca.

- ➢ Usa uma variável temporária para trocar os ponteiros das listas de vagões.
- ➢ Simples e eficaz: apenas os ponteiros são trocados, sem realocação de memória

12) printaLista: Percorre a lista de trens e seus vagões, imprimindo todas as informações.

Propósito: Permite visualizar o estado completo do sistema, útil para testes e depuração.

Construção:

- ➢ Percorre todos os trens da lista.
- ➢ Para cada trem, percorre e imprime todos os seus vagões.
- ➢ Utiliza printf para mostrar o nome do trem, o tipo de carga e a capacidade dos vagões.
- ➢ Serve como visualizador completo do sistema atual.

Exemplos de uso:

Seja a seguinte função main com as funções:

- adicionarTremOrdenado;
- adicionarVagaoOrdenado;
- printaLista;

```c
#include <stdio.h>
#include "trabalho.h"

int main() {
    NoTrens *listaTrens;

    crialistaTrem(&listaTrens);

    adicionarTremOrdenado(&listaTrens, "TremA");
    adicionarTremOrdenado(&listaTrens, "TremB");
    adicionarTremOrdenado(&listaTrens, "TremC");

    adicionarVagaoOrdenado(&listaTrens, "TremA", "Carvao", 30);
    adicionarVagaoOrdenado(&listaTrens, "TremA", "Ferro", 50);
    adicionarVagaoOrdenado(&listaTrens, "TremA", "Madeira", 40);

    adicionarVagaoOrdenado(&listaTrens, "TremB", "Passageiros", 200);
    adicionarVagaoOrdenado(&listaTrens, "TremB", "Bagagem", 20);

    adicionarVagaoOrdenado(&listaTrens, "TremC", "Passageiros", 100);
    adicionarVagaoOrdenado(&listaTrens, "TremC", "Alimentos", 15);

    printf("Lista de trens e seus respectivos vagoes:\n\n");
    printaLista(&listaTrens);

    return 0;
}
```

Inserimos 3 trens de forma ordenada, os trens “tremA”, “tremB” e “tremC”;

Preenchemos os vagões do tremA de forma ordenada, com vagões para materiais de construção;

Preenchemos os vagões do tremB também de forma ordenada, com passageiros e as suas bagagens;

Por fim, os vagoes do tremC também foram inseridos ordenados, com passageiros e alimentos (como grãos);

Após compilar e executar o programa realizado por essa main, teremos o seguinte resultado no terminal:

Conclusão:

Durante a implementação do sistema de gerenciamento de trens e vagões, enfrentamos diversos desafios em relação à manipulação de listas duplamente encadeadas,

principalmente nas funções de inserções ordenadas. Ter cuidado crítico com os ajustes dos ponteiros é essencial para evitar problemas relacionados a vazamento de memória. Outro desafio enfrentado foi a necessidade de manter a legibilidade e organização do código, já que devemos prestar muita atenção às condições de parada das funções e aos casos especiais (como, por exemplo, receber uma lista vazia).

Principais lições aprendidas:

- A importância de testar cada função individualmente;
- A utilidade de utilizar funções auxiliares;
- Ter cuidado com a alocação e liberação de memória do programa.