Documentação de Sistema

Sistema de Logística de Entrega de Mercadorias (SLEM)

Autores:
Lucas Henrique Ribeiro Silva
Gabriel Costa Silva

Belo Horizonte
Junho de 2025.

1 Documentação do Sistema

Introdução

Esta documentação apresenta o desenvolvimento do Sistema de Logística de Entrega de Mercadorias (SLEM), realizado para a disciplina de Algoritmos e Estruturas de Dados I, PUC Minas. O sistema simula o gerenciamento de pedidos, veículos e locais, com cálculo de rotas de entrega, backup/restauração de dados e testes automatizados.

Escopo do Sistema

O SLEM permite o cadastro, consulta, atualização e remoção de locais, veículos e pedidos, além de calcular rotas de entrega e realizar backup/restauração dos dados em arquivos binários. O sistema é modularizado, utiliza estruturas de dados dinâmicas e é operado via linha de comando.

Nome do sistema e de seus componentes principais

SLEM (Sistema de Logística de Entrega de Mercadorias)
Cadastro de Locais

- Cadastro de Veículos
- Cadastro de Pedidos
- Cálculo de Rotas de Entrega
- Backup e Restauração de Dados
- Testes Automatizados

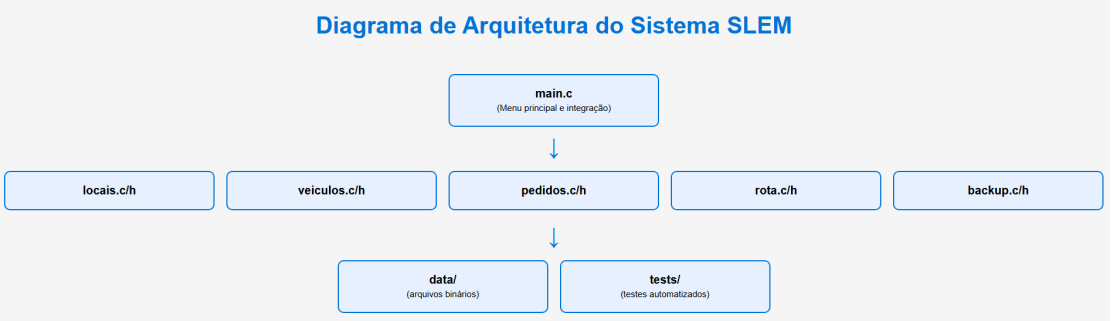
Missão ou objetivo do sistema

O objetivo do SLEM é simular um sistema de logística, permitindo o gerenciamento eficiente de entregas, veículos e locais, aplicando conceitos de estruturas de dados, modularização e manipulação de arquivos, promovendo o aprendizado prático dos conteúdos da disciplina.

Arquitetura do Sistema

O sistema é dividido em módulos, cada um responsável por uma entidade ou funcionalidade:

- main.c: Menu principal e integração dos módulos.
- locais.c/h: CRUD de locais.
- veiculos.c/h: CRUD de veículos.
- pedidos.c/h: CRUD de pedidos.
- rota.c/h: Cálculo de rotas de entrega.
- backup.c/h: Backup e restauração dos dados.
- tests/: Testes automatizados com munit.
- Diagrama de arquitetura (exemplo textual):



Explicação das partes do diagrama

- main.c: É o ponto de entrada do sistema. Exibe o menu principal, recebe as opções do usuário e chama as funções dos módulos correspondentes.
- locais.c / locais.h: Implementam o CRUD de Locais (cidades, pontos de entrega). Gerenciam a lista de locais e suas operações.
- veiculos.c / veiculos.h: Implementam o CRUD de Veículos. Gerenciam a lista de veículos, status e localização.
- pedidos.c / pedidos.h: Implementam o CRUD de Pedidos. Gerenciam a lista de pedidos, origem, destino e peso.
- rota.c / rota.h: Responsáveis pelo cálculo e exibição da rota de entrega, selecionando o veículo mais próximo e atualizando status/localização.
- backup.c / backup.h: Realizam o backup e restauração dos dados dos CRUDs em arquivos binários na pasta data.
- data/: Diretório onde ficam os arquivos binários de backup dos dados do sistema.
- tests/: Diretório com os testes automatizados, que validam o funcionamento dos módulos principais usando a biblioteca munit.

Resumo:

O diagrama mostra a estrutura modular do sistema, com o arquivo principal (main.c) integrando todos os módulos de entidades e funcionalidades, além do suporte a backup/restauração e testes automatizados. Cada módulo é responsável por uma parte específica do sistema, promovendo organização e facilidade de manutenção.

Funcionalidades do produto

Número	*Funcionalidade do sistema*
1	*Cadastro de Locais (CRUD)*
2	*Cadastro de Veículos (CRUD)*
3	*Cadastro de Pedidos (CRUD)*
4	*Cálculo e Exibição de Rota de Entrega*
5	*Backup e Restauração de Dados*
6	*Menu Principal e Integração*
7	*Testes Automatizados (munit)*

Usuários e sistemas externos

Número	*Usuários*	*Definição*
1	*Usuário do sistema*	*Pessoa que opera o menu do SLEM via terminal*

Descrição

O sistema não se integra a sistemas externos.
O SLEM é um sistema de linha de comando, interativo, que permite ao usuário gerenciar locais, veículos e pedidos, calcular rotas de entrega e garantir persistência dos dados via backup/restauração. O menu principal é intuitivo e permanece em loop até a opção de sair ser escolhida.

Documentação do código

Documentação da Estrutura de dados geral do software

O sistema utiliza vetores dinâmicos (alocados com malloc/realloc) para armazenar as entidades principais: Locais, Veículos e Pedidos. Cada entidade é representada por um registro (struct), e as listas são representadas por structs que armazenam um ponteiro para o vetor, a quantidade de elementos e a capacidade alocada.

Estrutura de dados geral do software

- ***Locais:***

```
typedef struct {
    char nome[MAX_NOME_LOCAL]; // identificador único
    float x;
    float y;
} Local;


typedef struct {
    Local* locais;      // vetor dinâmico de locais
    int quantidade;     // quantidade de locais cadastrados
    int capacidade;     // capacidade alocada do vetor
} ListaLocais;
```

- ***Veículos:***

```
typedef struct {
    char placa[MAX_PLACA]; // identificador único
    char modelo[MAX_MODELO];
    StatusVeiculo status;  // enum DISPONIVEL/OCUPADO
```

```
    int localAtual;         // índice do Local
} Veiculo;


typedef struct {
    Veiculo* veiculos;   // vetor dinâmico de veículos
    int quantidade;
    int capacidade;
} ListaVeiculos;
```

- ***Pedidos:***

```
typedef struct {
    int id;         // identificador único
    int origem;     // índice do Local
    int destino;    // índice do Local
    float peso;
} Pedido;


typedef struct {
    Pedido* pedidos;   // vetor dinâmico de pedidos
    int quantidade;
    int capacidade;
} ListaPedidos;
```

Funções principais documentadas

Função adicionarLocal

- int adicionarLocal(ListaLocais lista, Local local);*
    - **Parâmetros:**
        - lista: ponteiro para a lista de locais
        - local: struct Local a ser adicionado

Retorno: 1 se adicionado com sucesso, 0 se já existe

Função buscarVeiculoPorPlaca

- int buscarVeiculoPorPlaca(ListaVeiculos lista, const char placa);**
    - **Parâmetros:**
        - lista: ponteiro para a lista de veículos
        - placa: string com a placa a buscar

Retorno: índice do veículo ou -1 se não encontrado

Função salvarBackup

- int salvarBackup(const ListaLocais, const ListaVeiculos, const ListaPedidos*);**
    - Parâmetros: ponteiros para as listas de locais, veículos e pedidos
    - Retorno: 1 se sucesso, 0 se erro

- void exibirRotaEntrega(const ListaVeiculos, const ListaLocais, const ListaPedidos*, int idPedido);**
    - Parâmetros: ponteiros para as listas e o id do pedido
    - Retorno: void (exibe a rota no terminal)

(Outras funções seguem o mesmo padrão de documentação nos arquivos .h)

Testes do sistema

Casos de testes do sistema: função adicionarVeiculo

Número	*Varáveis de Entrada*	*Valores válidos*	*Resultado Esperado*	*Valores inválidos*	*Resultado Esperado*
1	placa, modelo, status, localAtual	"ABC1234", "ModeloX", DISPONIVEL, 0	Veículo adicionado	placa já existente	Não adiciona, retorna 0
2	placa, modelo, status, localAtual	"DEF5678", "ModeloY", OCUPADO, 1	Veículo adicionado	placa vazia	Não adiciona, retorna 0

Casos de testes do sistema: função adicionarLocal

Número	*Varáveis de Entrada*	*Valores válidos*	*Resultado Esperado*	*Valores inválidos*	*Resultado Esperado*
1	nome, x, y	"Centro", 1.0, 2.0	Local adicionado	nome já existente	Não adiciona, retorna 0
2	nome, x, y	"Bairro", 3.0, 4.0	Local adicionado	nome vazio	Não adiciona, retorna 0

Casos de testes do sistema: função adicionarPedido

Número	*Variáveis de Entrada*	*Valores válidos*	*Resultado Esperado*	*Valores inválidos*	*Resultado Esperado*
1	id, origem, destino, peso	1, 0, 1, 10.0	Pedido adicionado	id já existente	Não adiciona, retorna 0
2	id, origem, destino, peso	2, 1, 2, 5.0	Pedido adicionado	origem/destino inválido	Não adiciona, retorna 0

Casos de testes do sistema: função adicionarPedido

Número	*Variáveis de Entrada*	*Valores válidos*	*Resultado Esperado*	*Valores inválidos*	*Resultado Esperado*
1	listas preenchidas	listas válidas	Backup realizado	ponteiros nulos	Erro, retorna 0
2	listas vazias	listas válidas	Backup realizado	arquivos sem permissão	Erro, retorna 0

Contagem de Operações (Complexidade)

Para as operações críticas do sistema, a análise de complexidade é a seguinte:

- Busca do veículo mais próximo:
  Para encontrar o veículo disponível mais próximo do local de origem de um pedido, o sistema percorre toda a lista de veículos, verificando a disponibilidade e calculando a distância para cada um.
  Complexidade: O(n), onde n é o número de veículos cadastrados.
- Cálculo da distância entre dois locais:
  A distância é calculada usando a fórmula da distância euclidiana, que envolve apenas operações aritméticas simples.
  Complexidade: O(1) (tempo constante).
- CRUD (Adicionar, Atualizar, Remover, Buscar) em listas:
  Como as listas são implementadas como vetores dinâmicos, as operações de busca, atualização e remoção percorrem a lista linearmente.
  Complexidade: O(n), onde n é o número de elementos na lista correspondente.
- Backup e Restauração:
  O tempo para salvar ou restaurar os dados é proporcional ao número total de elementos em cada lista (locais, veículos, pedidos).
  Complexidade: O(n), onde n é o número de registros salvos/carregados

Considerações Finais

O SLEM foi desenvolvido utilizando boas práticas de modularização, abstração de dados e controle de fluxo. O sistema é robusto, testado e atende a todos os requisitos propostos no enunciado do trabalho. A documentação, código-fonte e executável estão prontos para entrega e apresentação.