PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS – PUC MINAS

Curso de Engenharia de Software

João Eduardo Soares Moreira
Rafael Nagem Volpini

# SISTEMA DE LOGÍSTICA DE ENTREGA DE MERCADORIAS (SLEM):
Projeto Interdisciplinar das disciplinas Algoritmos e Estruturas de Dados I e Fundamentos de Engenharia de Software

Belo Horizonte
2025

## 1 INTRODUÇÃO

O presente documento descreve o desenvolvimento do Sistema de Logística de Entrega de Mercadorias (SLEM), realizado como atividade interdisciplinar das disciplinas Algoritmos e Estruturas de Dados I (AEDS I) e Fundamentos de Engenharia de Software. O sistema simula o funcionamento básico de uma empresa de logística, incluindo a gestão de locais, veículos e pedidos de entrega.

O objetivo é aplicar conceitos fundamentais da programação estruturada, modularização de código, abstração de dados com structs, persistência de dados com arquivos binários e a lógica de roteamento e simulação. Ao mesmo tempo, o projeto reflete os princípios de Engenharia de Software com foco na análise de requisitos, organização modular e elaboração de documentação técnica.

## 2 OBJETIVOS

O projeto tem como objetivo principal desenvolver um sistema funcional e testável que gerencie uma operação logística simplificada, utilizando a linguagem C++. Os objetivos específicos incluem:

- Implementar estruturas de dados para representar locais, veículos e pedidos;
- Criar menus interativos para cadastro, atualização e exclusão dos registros;
- Simular entregas com lógica de alocação de veículos e cálculo de distâncias;
- Integrar os módulos com persistência via arquivos binários;
- Criar scripts automatizados para geração de dados e execução;
- Aplicar os princípios de desenvolvimento modular e documentação técnica.

## 3 FUNDAMENTAÇÃO TEÓRICA

As disciplinas envolvidas neste trabalho fornecem a base teórica para sua execução:

- Algoritmos e Estruturas de Dados I: fornece os conceitos de variáveis compostas (structs), vetores, funções e manipulação de arquivos, essenciais para o armazenamento e recuperação de dados.
- Fundamentos de Engenharia de Software: contribui com os princípios de análise de requisitos, modelagem modular, documentação e testes.

O uso de structs permite modelar as entidades principais do sistema. A modularização do código garante organização e manutenção facilitada. A simulação de entregas reforça a aplicação prática de algoritmos simples com impacto real na lógica do negócio.

## 4 DOCUMENTAÇÃO DA FUNÇÃO

**pedidos.cpp**

/**

* Verifica se um veículo está em uso em algum pedido com status EM_ENTREGA.

*

* @param idVeiculo ID do veículo a ser verificado.

* @param pedidos Vetor de pedidos registrados.

* @param qtdPedidos Quantidade total de pedidos no vetor.

* @param idIgnorar Índice do pedido que deve ser ignorado na verificação.

* @return 1 se o veículo estiver em uso, 0 caso contrário.

*/

int veiculoEmUso(int idVeiculo, const Pedido pedidos[], int qtdPedidos, int idIgnorar)

**veiculo.cpp**

/**

 * Procura um veículo no vetor a partir da placa.

 *

 * @param veiculos Vetor de veículos.

 * @param qtdVeiculos Quantidade de veículos no vetor.

 * @param placa Placa do veículo a ser encontrado.

 * @return Índice do veículo se encontrado, -1 caso contrário.

 */

int encontrarVeiculoPorPlaca(const Veiculo veiculos[], int qtdVeiculos, const char* placa);

**sistema.cpp**

```
/**
 * Calcula a distância euclidiana entre dois locais.
 *
 * @param origem Local de origem.
 * @param destino Local de destino.
 * @return Distância entre os dois locais. Retorna -1.0 em caso de valores inválidos.
 */
float calcularDistancia(Local origem, Local destino);
```

**locais.cpp**

```
/**
 * Cadastra um novo local no vetor de locais, com validação de nome e coordenadas.
 *
 * @param locais Vetor de locais.
 * @param quantidade Ponteiro para a quantidade atual de locais.
 */
void cadastrarLocal(Local locais[], int *quantidade);
```

## 5 METODOLOGIA

O projeto foi estruturado em fases de desenvolvimento incremental:

1. Modelagem das estruturas de dados (Local, Veículo, Pedido);
2. Implementação dos CRUDs individuais com validações;
3. Integração entre módulos e lógica de roteamento;
4. Criação de scripts para geração de dados realistas;
5. Elaboração de Makefile para facilitar compilação e testes;
6. Redação da documentação técnica.

Cada módulo (.cpp e .h) foi implementado com funções específicas e chamadas a partir de menus interativos. A simulação de entregas foi desenvolvida com duas etapas: iniciar e finalizar entrega, atualizando status e posições dos veículos.

## 5 DESCRIÇÃO DO SISTEMA

O sistema desenvolvido é composto por três entidades principais: Locais, Veículos e Pedidos. As funcionalidades básicas implementadas permitem a criação, visualização, edição e remoção dessas entidades, além de executar o fluxo de uma entrega de forma realista.

Cada entidade possui atributos definidos conforme a proposta. As estruturas são manipuladas em memória e persistidas em arquivos binários. Os menus são implementados de forma interativa com feedback ao usuário, além de validações em tempo de execução.

O fluxo de entrega foi dividido em duas etapas: iniciar entrega (aloca veículo e define status como EM_ENTREGA) e finalizar entrega (atualiza o status para ENTREGUE e desloca o veículo para o destino).

## 6 ANÁLISE DE COMPLEXIDADE

A principal operação de custo relevante no sistema é a busca do veículo mais próximo para iniciar uma entrega. Essa busca percorre todos os veículos ativos e disponíveis para determinar aquele com menor distância euclidiana em relação ao ponto de origem da entrega.

Como não foi utilizada nenhuma estrutura de dados especializada para otimização (ex: árvore k-d), o tempo de execução dessa operação é linear:

- Complexidade: O(n), onde n é o número de veículos disponíveis no sistema.

As demais operações (busca por índice, atualizações diretas e persistência) possuem complexidade constante ou linear, com impacto pequeno na execução geral.

## 7 RESULTADOS OBTIDOS

Ao final do desenvolvimento, o sistema foi capaz de:

- Gerenciar 20 locais distintos de Belo Horizonte;
- Registrar 30 veículos realistas com diferentes modelos e status;
- Simular 15 pedidos com diferentes combinações de origem, destino, peso e status;
- Realizar entregas simuladas com alocação inteligente de veículos;

- Atualizar a posição dos veículos após a entrega;
- Executar todas as funcionalidades com persistência em arquivos binários;
- Compilar e executar com facilidade via Makefile;
- Gerar dados de exemplo automaticamente para testes.

## 8 CASOS DE TESTE

### pedidos.cpp

| Caso | idVeiculo | Descrição do pedido | qtdPedidos | idIgnorar | Saída Esperada |
|---|---|---|---|---|---|
| 1 | 2 | status = EM_ENTREGA | 3 | -1 | 1 |
| 2 | 2 | status = PENDENTE | 3 | -1 | 0 |
| 3 | 1 | status = EM_ENTREGA, mas ignorado | 3 | 2 | 0 |
| 4 | -1 | idVeiculo inválido | 3 | -1 | 0 |
| 5 | 2 | lista de pedidos vazia | 0 | -1 | 0 |

### veiculo.cpp

| Caso | placa | Descrição do vetor de veículos | qtdVeiculos | placa entrada | Saída Esperada |
|---|---|---|---|---|---|
| 1 | "ABC1234" | placa presente na posição 0 | 3 | "ABC1234" | 0 |
| 2 | "ZZZ9999" | placa inexistente | 3 | "ZZZ9999" | -1 |
| 3 | NULL | entrada nula (placa inválida) | 3 | NULL | -1 |
| 4 | "ABC1234" | lista de veículos vazia | 0 | "ABC1234" | -1 |
| 5 | "abc1234" | case-sensitive (placa existe mas com caixa) | 3 | "abc1234" | -1 |

## sistema.cpp

| Caso | origem.x | origem.y | destino.x | destino.y | Descrição | Saída Esperada |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 3 | 4 | Distância clássica 3-4-5 | 5.0 |
| 2 | 1 | 1 | 1 | 1 | Mesma posição | 0.0 |
| 3 | -1 | -1 | 1 | 1 | Distância entre quadrantes | 2.828... |
| 4 | 10 | 5 | 5 | 1 | Diferença nos dois eixos | 6.403... |
| 5 | 0 | 0 | -3 | -4 | Espelhado do caso 1 | 5.0 |

## locais.cpp

| Caso | Estado Inicial | Entrada Simulada | Esperado | Saída Esperada |
|---|---|---|---|---|
| 1 | quantidade = 0 | nome = "BH", x = 1.1, y = 2.2 | Local adicionado com sucesso | quantidade = 1 |
| 2 | quantidade = MAX | - | Limite atingido, não cadastra | quantidade igual |
| 3 | nome = "" | string vazia | Rejeita, pede nova entrada | não altera |
| 4 | nome = " " | só espaços | Rejeita, pede nova entrada | não altera |
| 5 | coord X = "abc" | entrada inválida (texto) | Rejeita, pede nova entrada | não altera |

## 9 RELATÓRIO DE EXECUÇÃO DOS TESTES

### pedidos.cpp

| Caso | Entrada | Saída Esperada | Saída Real | Resultado |
|---|---|---|---|---|
| 1 | idVeiculo = 2, status = EM_ENTREGA | 1 | 1 | ✅ Passou |
| 2 | idVeiculo = 2, status = PENDENTE | 0 | 0 | ✅ Passou |
| 3 | idVeiculo = 1, ignorando índice 2 | 0 | 0 | ✅ Passou |
| 4 | idVeiculo = -1 (inválido) | 0 | 0 | ✅ Passou |
| 5 | lista de pedidos vazia | 0 | 0 | ✅ Passou |

### veiculo.cpp

| Caso | Entrada | Saída Esperada | Saída Real | Resultado |
|---|---|---|---|---|
| 1 | placa="ABC1234" | 0 | 0 | ✅ Passou |
| 2 | placa="ZZZ9999" | -1 | -1 | ✅ Passou |
| 3 | placa=NULL | -1 | -1 | ✅ Passou |
| 4 | qtdVeiculos=0 | -1 | -1 | ✅ Passou |
| 5 | placa="abc1234" (case-sensitive) | -1 | -1 | ✅ Passou |

**sistema.cpp**

| Caso | Entrada | Saída Esperada | Saída Real | Resultado |
|---|---|---|---|---|
| 1 | origem(0,0), destino(3,4) | 5.0 | 5.0 | ✅ Passou |
| 2 | origem(1,1), destino(1,1) | 0.0 | 0.0 | ✅ Passou |
| 3 | origem(-1,-1), destino(1,1) | 2.828... | ≈2.828 | ✅ Passou |
| 4 | origem(10,5), destino(5,1) | 6.403... | ≈6.403 | ✅ Passou |
| 5 | origem(0,0), destino(-3,-4) | 5.0 | 5.0 | ✅ Passou |

**locais.cpp**

| Caso | Entrada | Saída Esperada | Saída Real | Resultado |
|---|---|---|---|---|
| 1 | " " | 1 | 1 | ✅ Passou |
| 2 | "" | 1 | 1 | ✅ Passou |
| 3 | "BH" | 0 | 0 | ✅ Passou |
| 4 | " SP" | 0 | 0 | ✅ Passou |

## 10 CONCLUSÃO

O Sistema de Logística de Entrega de Mercadorias (SLEM) atingiu todos os objetivos propostos, mostrando a eficácia da aplicação de estruturas de dados, lógica algorítmica e conceitos fundamentais da engenharia de software. Além da codificação funcional, o projeto envolveu documentação adequada, testes com dados reais e organização de diretórios.

Esse trabalho demonstrou a capacidade de integrar diferentes conhecimentos de forma prática, fortalecendo o domínio das tecnologias utilizadas e a habilidade de estruturar soluções técnicas em equipe.