PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE
Bacharelado em Engenharia de Software

Nome dos integrantes do grupo:

-Bernardo Demaria Santos;

-Marcelo Esteves Bernardi;

-Raphael Thierry Coelho;

-Stêvão Guadanini Diniz.

Nome do sistema: LogiTrack

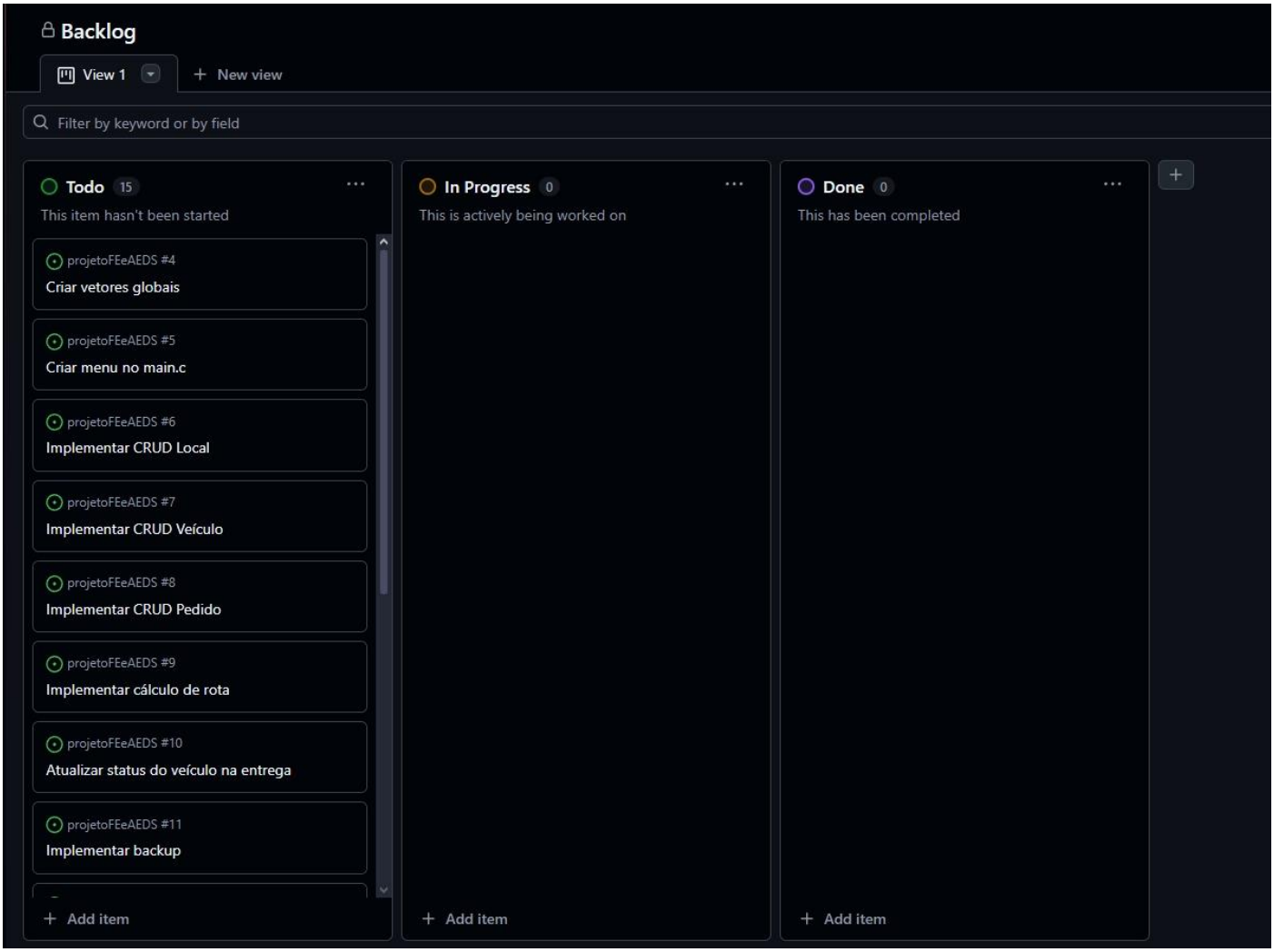
Apresentação:

O programa tem como objetivo simular a gestão de entregas, permitindo o cadastro e controle de locais, veículos e pedidos. O sistema calcula rotas com base na distância entre pontos, atualiza o status dos veículos durante a entrega e realiza backup e restauração dos dados em arquivos binários, aplicando os principais conceitos da disciplina de Algoritmos e Estruturas de Dados I.

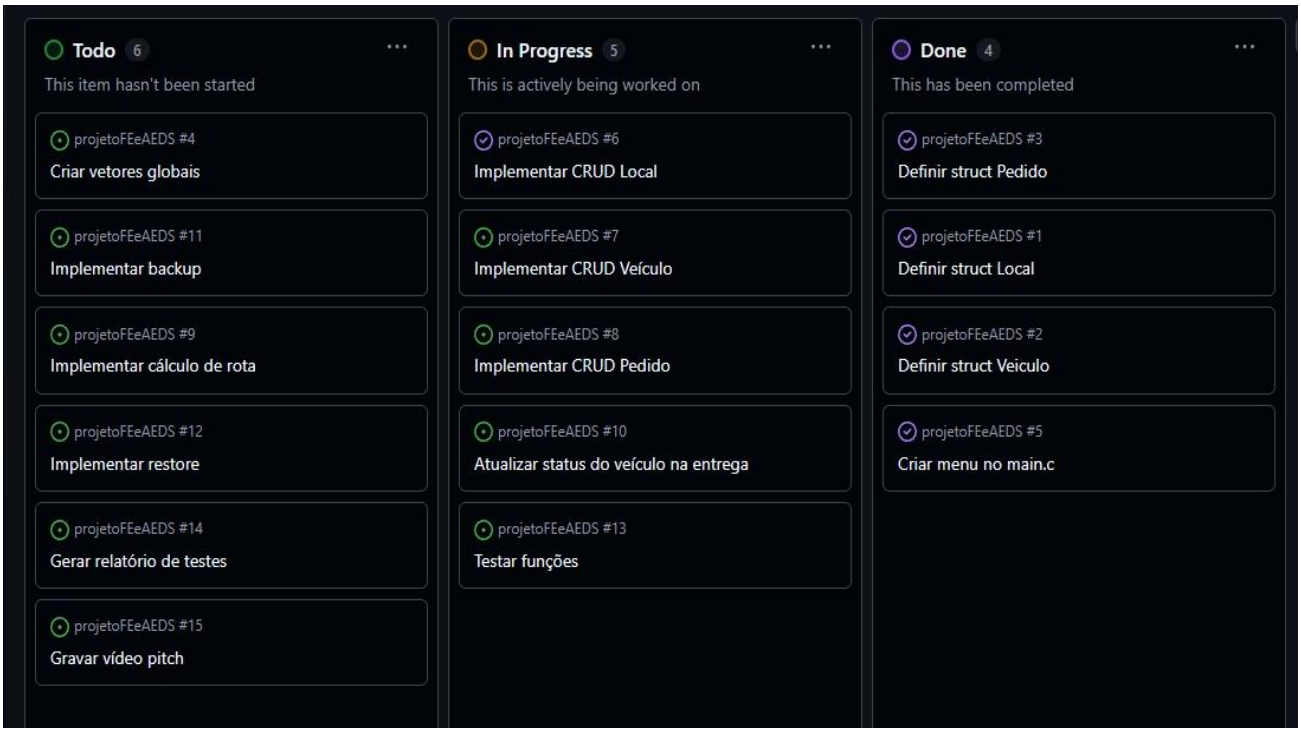
Backlog do produto:

Apresente o backlog do seu produto a cada semana.

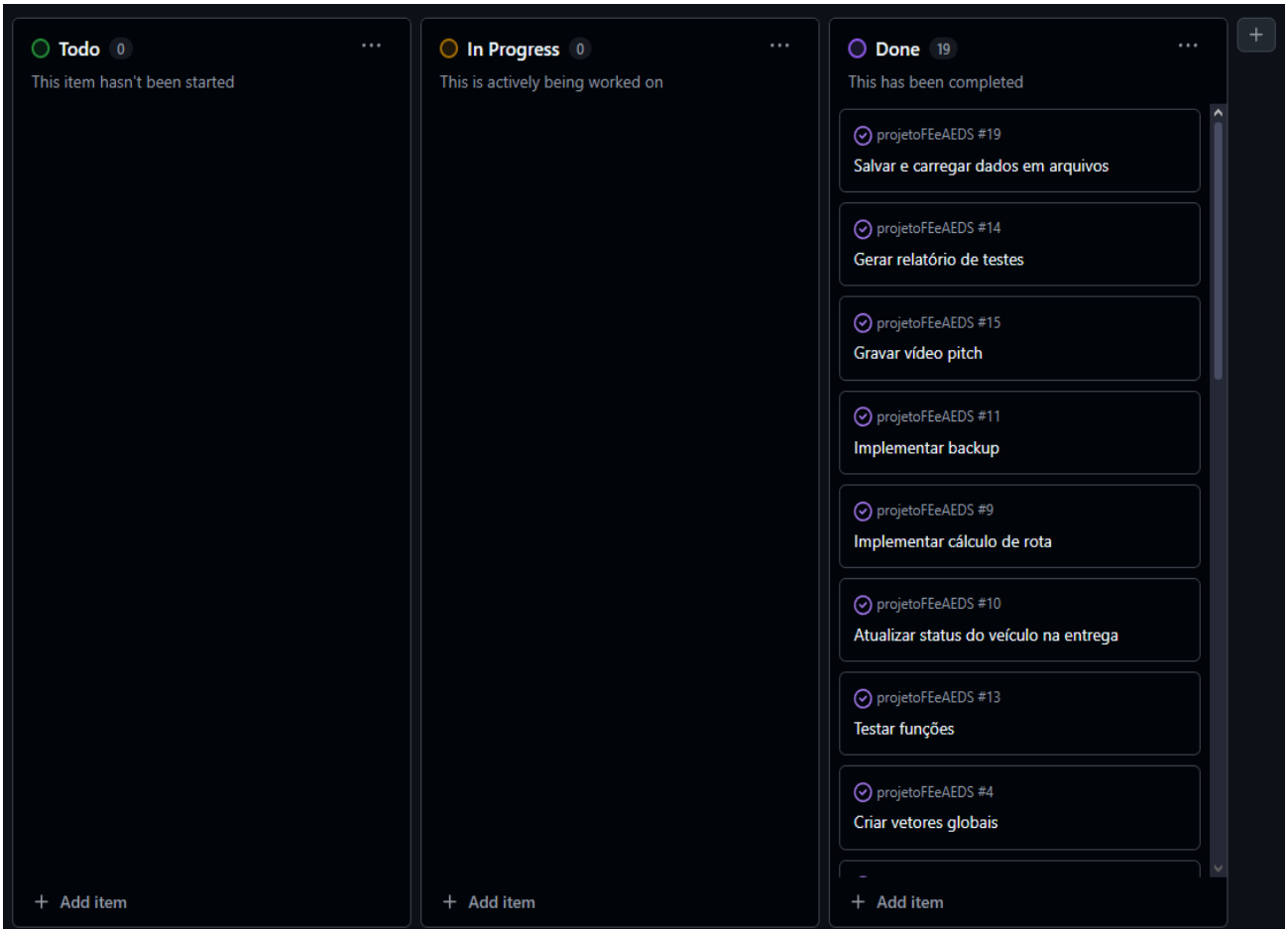
A Figura apresenta o backlog com as tarefas a serem feitas, divididas em issues no github para uma melhor organização. E o progresso do trabalho durante as sprints.



Fonte: Elaborado pelo autor



Fonte: Elaborado pelo autor



Fonte: Elaborado pelo autor

Lista de assinaturas das funções e parâmetros

Explicação da estrutura de dados principal do programa.

Apresentação das assinaturas das funções

As funções e parâmetros utilizados no programa foram:

1. void cadastrarLocal(Local locais[], int *numLocais)
Função responsável por cadastrar um novo local no vetor locais. Recebe como parâmetros o vetor de locais e um ponteiro para a variável que armazena o número atual de locais cadastrados (numLocais). A função solicita ao usuário o nome do local e suas coordenadas X e Y. Em seguida, armazena essas informações na próxima posição disponível do vetor. Caso o número máximo de locais (MAX_LOCAIS) já tenha sido atingido, a função emite uma mensagem de erro e retorna sem cadastrar.
Não possui valor de retorno.

---

2. void listarLocais(Local locais[], int numLocais)
Função que exibe todos os locais cadastrados. Recebe como parâmetros o vetor locais e o número total

de locais já cadastrados (numLocais).
A função percorre o vetor de locais e imprime o ID, o nome e as coordenadas (X, Y) de cada local na tela.
Não possui valor de retorno.

---

3. void atualizarLocal(Local locais[], int numLocais)
Função utilizada para atualizar os dados de um local já existente. Recebe como parâmetros o vetor de locais e o número total de locais cadastrados (numLocais).
O usuário informa o ID do local que deseja atualizar. Se o ID for inválido (menor que 0 ou maior ou igual ao número de locais), a função exibe uma mensagem de erro. Caso contrário, solicita novos dados (nome e coordenadas) e atualiza o local correspondente no vetor.
Não possui valor de retorno.

---

4. void removerLocal(Local locais[], int *numLocais)
Função que realiza a remoção de um local com base em seu ID. Recebe como parâmetros o vetor de locais e um ponteiro para o número atual de locais cadastrados (numLocais).
O usuário fornece o ID do local a ser removido. A função verifica se o ID é válido; se não for, exibe uma mensagem de erro. Se for válido, desloca todos os elementos subsequentes no vetor uma posição para trás, sobrescrevendo o local removido, e decrementa o total de locais cadastrados.
Não possui valor de retorno.

---

5. void cadastrarPedido(Pedido pedidos[], int *numPedidos)
Função para cadastrar um novo pedido. Recebe como parâmetros o vetor pedidos e um ponteiro para o total atual de pedidos cadastrados (numPedidos).
A função atribui um ID automático ao pedido com base na posição atual do vetor, solicita ao usuário os IDs dos locais de origem e destino, e o peso do pedido. Em seguida, armazena os dados e incrementa o número de pedidos. Caso o limite máximo (MAX_PEDIDOS) seja atingido, exibe uma mensagem e retorna sem cadastrar.
Não possui valor de retorno.

---

6. void listarPedidos(Pedido pedidos[], int numPedidos)
Função que imprime todos os pedidos cadastrados. Recebe como parâmetros o vetor pedidos e o número total de pedidos (numPedidos).
A função percorre o vetor e exibe, para cada pedido, seu ID, local de origem, local de destino e peso.
Não possui valor de retorno.

---

7. void atualizarPedido(Pedido pedidos[], int numPedidos)
Função utilizada para atualizar um pedido existente. Recebe como parâmetros o vetor pedidos e o número total de pedidos cadastrados (numPedidos).
O usuário informa o ID do pedido a ser atualizado. Se o ID for inválido, a função emite uma mensagem de erro. Caso contrário, solicita os novos dados (origem, destino e peso) e atualiza o pedido correspondente.
Não possui valor de retorno.

---

8. void removerPedido(Pedido pedidos[], int *numPedidos)
Função que remove um pedido do vetor. Recebe como parâmetros o vetor pedidos e um ponteiro para a quantidade de pedidos cadastrados (numPedidos).
O usuário fornece o ID do pedido a ser removido. Se for inválido, a função exibe uma mensagem. Se for válido, desloca os elementos subsequentes no vetor para sobrescrever o pedido removido e decrementa o total de pedidos.
Não possui valor de retorno.

---

9. void cadastrarVeiculo(Veiculo veiculos[], int *numVeiculos)
Função utilizada para cadastrar um novo veículo. Recebe como parâmetros o vetor veiculos e um ponteiro para o número atual de veículos cadastrados (numVeiculos).
A função solicita ao usuário os dados do veículo (placa, modelo e ID do local atual) e define seu status como **disponível** (0). Em seguida, armazena os dados na próxima posição do vetor e incrementa o número de veículos. Caso o limite (MAX_VEICULOS) tenha sido atingido, a função informa e encerra sem cadastrar.
Não possui valor de retorno.

---

10. void listarVeiculos(Veiculo veiculos[], int numVeiculos)
Função que exibe todos os veículos cadastrados. Recebe como parâmetros o vetor veiculos e a quantidade de veículos existentes (numVeiculos).
A função percorre o vetor e imprime, para cada veículo, seu índice, placa, modelo, status (disponível ou ocupado) e o ID do local onde está.
Não possui valor de retorno.

---

11. void atualizarVeiculo(Veiculo veiculos[], int numVeiculos)
Função para atualizar os dados de um veículo já cadastrado. Recebe como parâmetros o vetor veiculos e o número de veículos cadastrados (numVeiculos).
O usuário informa o índice do veículo a ser atualizado. A função valida o índice e, caso seja válido, solicita novos dados (placa, modelo, status e ID do local atual) e os atualiza no vetor.
Não possui valor de retorno.

---

12. void removerVeiculo(Veiculo veiculos[], int *numVeiculos)
Função que realiza a remoção de um veículo com base no índice informado. Recebe como parâmetros o vetor veiculos e um ponteiro para o número de veículos cadastrados (numVeiculos).
Após validar o índice fornecido, a função remove o veículo deslocando os elementos subsequentes uma posição para trás e decrementa o número total de veículos.
Não possui valor de retorno.

---

13. void associarPedidoRotaVeiculo(Pedido pedidos[], int numPedidos, Local locais[], int numLocais, Veiculo veiculos[], int numVeiculos)
Função responsável por associar o **último pedido cadastrado** ao **veículo disponível mais próximo do local de origem** do pedido. Recebe como parâmetros os vetores de pedidos, locais e veiculos, além da quantidade de cada um (numPedidos, numLocais, numVeiculos).

A função segue os seguintes passos:

- Verifica se há pedidos cadastrados.
- Recupera o **último pedido** e seus IDs de origem e destino.
- Verifica se os IDs de origem e destino são válidos.
- Busca o veículo disponível mais próximo do local de origem, utilizando a função buscarVeiculoMaisProximo.
- Calcula a distância da rota usando a função calcularDistancia.
- Exibe as informações da rota: veículo associado, origem, destino e distância.
- Atualiza o status do veículo para **ocupado** (1) e altera seu local atual para o local de destino do pedido.

A função **não possui valor de retorno**. Caso não haja veículos disponíveis ou os dados do pedido sejam inválidos, ela exibe mensagens apropriadas.

---

14. void finalizarEntrega(Veiculo veiculos[], int numVeiculos)
Função responsável por **finalizar a entrega de um pedido** e atualizar o status do veículo correspondente. Recebe como parâmetros o vetor veiculos e o número total de veículos cadastrados (numVeiculos).

O usuário deve informar o ID (índice) do veículo que está finalizando a entrega. A função verifica:

- Se o ID é válido (dentro do intervalo de veículos cadastrados),
- Se o veículo informado está realmente com status **ocupado** (status == 1).

Se os dados forem válidos, a função altera o status do veículo para **disponível** (status = 0) e exibe uma mensagem confirmando a finalização da entrega. Caso contrário, informa que o ID é inválido ou que o veículo não estava em entrega.
A função **não possui valor de retorno**.

---

15. void salvarLocais(Local locais[], int numLocais)
Função responsável por salvar os dados dos locais cadastrados em um **arquivo binário**. Recebe como parâmetros o vetor locais e a quantidade atual de locais cadastrados (numLocais).

A função abre o arquivo "locais.dat" em modo de escrita binária ("wb"). Se houver erro na abertura do arquivo, exibe uma mensagem e retorna. Caso contrário, ela grava:

- Primeiro, o valor de numLocais (quantidade de locais),
- Em seguida, os dados do vetor locais.

Após a escrita, o arquivo é fechado e uma mensagem de sucesso é exibida.
A função **não possui valor de retorno**.

---

16. void carregarLocais(Local locais[], int *numLocais)
Função responsável por **carregar os dados dos locais salvos** em um arquivo binário. Recebe como parâmetros o vetor locais e um ponteiro para a variável que armazena a quantidade de locais cadastrados (numLocais).

A função tenta abrir o arquivo "locais.dat" em modo de leitura binária ("rb").

- Se o arquivo não for encontrado, exibe uma mensagem informando que não há backup disponível e define *numLocais como 0.
- Se o arquivo for aberto com sucesso, a função lê primeiro o valor de numLocais e, em seguida, carrega os dados do vetor locais a partir do conteúdo gravado no arquivo.

Após o carregamento, o arquivo é fechado e uma mensagem de sucesso é exibida.
A função **não possui valor de retorno**.

---

17. void salvarVeiculos(Veiculo veiculos[], int numVeiculos)
Função responsável por **salvar os dados dos veículos cadastrados** em um **arquivo binário**. Recebe como parâmetros o vetor veiculos e o número total de veículos cadastrados (numVeiculos).

A função abre o arquivo "veiculos.dat" em modo de escrita binária ("wb").

- Se ocorrer erro na abertura do arquivo, exibe uma mensagem e encerra a função.
- Caso o arquivo seja aberto com sucesso, grava:
    - O valor de numVeiculos,
    - Os dados do vetor veiculos, com todos os veículos cadastrados.

Ao final, o arquivo é fechado e uma mensagem de sucesso é exibida.
A função **não possui valor de retorno**.

---

18. void carregarVeiculos(Veiculo veiculos[], int *numVeiculos)
Função responsável por **carregar os dados dos veículos** a partir de um **arquivo binário** previamente salvo. Recebe como parâmetros o vetor veiculos e um ponteiro para a variável que armazena a quantidade de veículos cadastrados (numVeiculos).

A função tenta abrir o arquivo "veiculos.dat" em modo de leitura binária ("rb").

- Se o arquivo não for encontrado, a função informa que **não há backup disponível** e zera o número de veículos.
- Caso o arquivo exista, a função lê:
    - O valor de numVeiculos,
    - Em seguida, os dados do vetor veiculos.

Por fim, fecha o arquivo e exibe uma mensagem indicando o carregamento bem-sucedido.
A função **não possui valor de retorno**.

---

19. void salvarPedidos(Pedido pedidos[], int numPedidos, const char* nomeArquivo)
Função responsável por **salvar os dados dos pedidos** em um **arquivo binário** cujo nome é fornecido como parâmetro. Recebe como parâmetros:

- o vetor pedidos com os pedidos cadastrados,

- o número total de pedidos (numPedidos),
- e uma string com o nome do arquivo (nomeArquivo).

A função tenta abrir o arquivo especificado em modo de escrita binária ("wb").

- Se não conseguir abrir o arquivo, exibe uma mensagem de erro e encerra a execução da função.
- Caso contrário, grava:
    - Primeiro o número total de pedidos,
    - Depois o conteúdo completo do vetor pedidos.

Ao final, fecha o arquivo.
A função **não possui valor de retorno**.

---

20. void carregarPedidos(Pedido pedidos[], int *numPedidos, const char* nomeArquivo)
Função responsável por **carregar os dados dos pedidos** a partir de um **arquivo binário**. Recebe como parâmetros:

- o vetor pedidos para armazenar os dados,
- um ponteiro para o número total de pedidos (numPedidos),
- e uma string com o nome do arquivo (nomeArquivo) de onde os dados serão lidos.

A função tenta abrir o arquivo indicado em modo de leitura binária ("rb").

- Se o arquivo não for encontrado, exibe uma mensagem informando a ausência do backup e define *numPedidos = 0.
- Se o arquivo for aberto com sucesso, a função lê:
    - o número de pedidos cadastrados,
    - os dados completos do vetor pedidos.

Por fim, fecha o arquivo.
A função **não possui valor de retorno**.

TESTES

Casos de teste do software:

Os casos de teste englobam todo o código, uma vez que as funções devem receber parâmetros que podem estar dentro de outras funções ou do código principal.

Entradas	Classes Válidas	Resultado Esperado	Classes Inválidas	Resultado Esperado
Opção: Número inteiro digitado pelo usuário	Números inteiros de 0 a 16	Função correspondente executada, ou saída se for 0	Números fora do intervalo (ex: -5, 17, 100), letras (ex: 'a'), símbolos (ex: '#')	Exibe “Opção inválida!” e retorna ao menu principal

---

Relatório de Execução de Testes

Entrada	Resultado Obtido	Aprovado?
Valor: 1	Cadastra local	Sim
Valor: 2	Lista locais	Sim
Valor: 10	Lista pedidos	Sim
Valor: 14	Finaliza entrega	Sim
Valor: 0	Encerra o programa com “Saindo...”	Sim
Valor: 17	Exibe “Opção inválida!”	Sim
Valor: -1	Exibe “Opção inválida!”	Sim
Valor: ‘a’	Falha (scanf não lê corretamente valores não numéricos)	Não*

Imagens Mostrando os testes automatizados executados com a biblioteca munit:

Alguns testes Executados:

MUNIT.H:

O código C do programa e suas funções incluindo a implementação automatizada dos casos de testes.

Código em C:

#include <stdio.h>

#include "../includes/local.h"

#include "../includes/veiculo.h"

```
#include "../includes/pedido.h"
#include "../includes/rota.h"
#include "../includes/arquivo.h"

#define MAX_LOCAIS 100
#define MAX_VEICULOS 100
#define MAX_PEDIDOS 100

Local locais[MAX_LOCAIS];
Veiculo veiculos[MAX_VEICULOS];
Pedido pedidos[MAX_PEDIDOS];

int numLocais = 0, numVeiculos = 0, numPedidos = 0;

void menu() {
   int opcao;
   do {
      printf("\n==== Sistema de Logistica de Entrega de Mercadorias ====");
      printf("\n1. Cadastrar Local");
      printf("\n2. Listar Locais");
      printf("\n3. Atualizar Local");
      printf("\n4. Remover Local");
      printf("\n5. Cadastrar Veiculo");
      printf("\n6. Listar Veiculos");
      printf("\n7. Atualizar Veiculo");
      printf("\n8. Remover Veiculo");
      printf("\n9. Cadastrar Pedido");
      printf("\n10. Listar Pedidos");
      printf("\n11. Atualizar Pedido");
      printf("\n12. Remover Pedido");
      printf("\n13. Associar pedido a rota e veiculo");
      printf("\n14. Finalizar entrega");
      printf("\n15. Salvar backup");
      printf("\n16. Carregar backup");
      printf("\n0. Sair");
      printf("\nEscolha: ");
      scanf("%d", &opcao);

      switch(opcao) {
         case 1: cadastrarLocal(locais, &numLocais); break;
         case 2: listarLocais(locais, numLocais); break;
         case 3: atualizarLocal(locais, numLocais); break;
         case 4: removerLocal(locais, &numLocais); break;
         case 5: cadastrarVeiculo(veiculos, &numVeiculos); break;
         case 6: listarVeiculos(veiculos, numVeiculos); break;
         case 7: atualizarVeiculo(veiculos, numVeiculos); break;
```

```
        case 8: removerVeiculo(veiculos, &numVeiculos); break;
        case 9: cadastrarPedido(pedidos, &numPedidos); break;
        case 10: listarPedidos(pedidos, numPedidos); break;
        case 11: atualizarPedido(pedidos, numPedidos); break;
        case 12: removerPedido(pedidos, &numPedidos); break;
        case 13: associarPedidoRotaVeiculo(pedidos, numPedidos, locais, numLocais, veiculos, numVeiculos); break;
        case 14: finalizarEntrega(veiculos, numVeiculos); break;
        case 15:
          salvarLocais(locais, numLocais);
          salvarVeiculos(veiculos, numVeiculos);
          salvarPedidos(pedidos, numPedidos, "pedidos.dat");
        break;
        case 16:
          carregarLocais(locais, &numLocais);
          carregarVeiculos(veiculos, &numVeiculos);
          carregarPedidos(pedidos, &numPedidos, "pedidos.dat");
        break;
        case 0: printf("Saindo...\n"); break;
        default: printf("Opção invalida!\n"); break;
      }
   } while (opcao != 0);
}

int main() {
   menu();
   return 0;
}
```

Documentação Técnica - Contagem de Operações na Função de Rota

Função Analisada: `associarPedidoRotaVeiculo`

Essa função é responsável por associar um pedido ao veículo mais próximo disponível e calcular a distância da entrega.

Contagem de Operações

Etapas Fixas (executadas uma vez):

- **Verificação de pedidos existentes**: 1 comparação
- **Atribuições e acessos a índices de arrays**: 5 operações

- **Validação de origem e destino do pedido**: 7 operações
- **Cálculo da distância entre origem e destino**: 6 operações
  - 2 subtrações
  - 2 multiplicações
  - 1 soma
  - 1 raiz quadrada
- **Atualização do status e posição do veículo**: 2 atribuições

Total fixo: **21 operações**

Etapas Variáveis (por veículo disponível):

A função `buscarVeiculoMaisProximo` realiza uma busca linear entre os veículos disponíveis. Para **cada veículo disponível**, são realizadas cerca de **10 operações**, incluindo:

- Verificação de status do veículo
- Cálculo da distância (mesma fórmula acima)
- Comparações e atualizações de menor distância

Total por veículo disponível: **≈ 10 operações**

Estimativa Total de Operações

Se houver `n` veículos disponíveis, o número total estimado de operações será:

Total ≈ 21 + 10 × n operações

Onde:
- `n` = quantidade de veículos disponíveis no momento da chamada.

Observações

- As operações de impressão (`printf`) não foram contabilizadas, pois não afetam diretamente o custo computacional.
- O número real pode variar levemente dependendo das condições de entrada, mas essa análise oferece uma boa estimativa de complexidade prática.

Vídeo Pitch do programa em funcionamento: https://youtu.be/oCDi-iE7Umw?si=zg_zeimgu8ysghr8