1. Introdução

Este relatório descreve o "Sistema de Gerenciamento de Estacionamento", um programa desenvolvido em linguagem C que simula um sistema básico para controle de veículos em um estacionamento. O projeto visa aplicar conceitos fundamentais da programação estruturada, incluindo controle de fluxo, laços de repetição, funções, ponteiros, estruturas, arranjos e manipulação de arquivos, conforme os requisitos estabelecidos para o projeto de Programação I.

O programa permite ao usuário gerenciar a entrada e saída de veículos, visualizar veículos estacionados, gerar relatórios financeiros e consultar o histórico completo de movimentações. A interface é interativa e baseada em terminal, com foco na usabilidade e robustez.

2. Funcionalidades Implementadas

O sistema oferece as seguintes funcionalidades principais, acessíveis através de um menu interativo:

- **2.1. Registrar Entrada de Veículo:**
  - Permite ao usuário informar a placa de um veículo que está entrando no estacionamento.
  - **Valida o formato da placa para seguir os padrões brasileiros:**
    - **Padrão Antigo:** `LLL-NNNN` (três letras, hífen, quatro números).
    - **Padrão Mercosul:** `LLLNLNN` (três letras, um número, uma letra, dois números).
  - Verifica se a capacidade máxima do estacionamento não foi atingida (definida por `MAX_PLACAS`).
  - Impede o registro de uma placa que já esteja ativa (veículo já estacionado).
  - Registra a placa, a data e hora de entrada (timestamp) e marca o veículo como "ativo" (estacionado).
  - Os dados do veículo são automaticamente salvos no arquivo `registros_completos.txt` após o registro.
- **2.2. Registrar Saída de Veículo:**
  - Exibe uma lista numerada de todos os veículos atualmente estacionados, permitindo ao usuário selecionar qual veículo está saindo.
  - Solicita confirmação do usuário antes de finalizar a saída.
  - Registra a data e hora de saída (timestamp).
  - Calcula o tempo total de permanência no estacionamento em minutos e segundos.
  - Calcula o valor a ser pago com base em uma tarifa por minuto (definida por `TARIFA_POR_MINUTO`) e aplica uma tarifa mínima (definida por `TARIFA_MINIMA`).
  - Marca o veículo como "inativo" (saiu do estacionamento) e armazena o valor pago.

    - Os dados atualizados do veículo são automaticamente salvos no arquivo `registros_completos.txt`.
- **2.3. Listar Veículos Estacionados:**
    - Exibe uma lista clara de todos os veículos que estão atualmente dentro do estacionamento.
    - Para cada veículo, mostra a placa e a data e hora de entrada.
    - Informa se não houver veículos estacionados no momento.
- **2.4. Gerar Relatório Financeiro:**
    - Apresenta um relatório consolidado dos valores arrecadados.
    - Lista as placas dos veículos que já saíram e o valor pago por cada um.
    - Calcula e exibe o "TOTAL ARRECADADO" geral.
    - Considera apenas os veículos que já registraram saída e possuem um valor pago.
- **2.5. Histórico de Veículos:**
    - Exibe um histórico completo de todos os veículos que passaram pelo estacionamento, incluindo os que ainda estão estacionados e os que já saíram.
    - Para cada veículo, mostra a placa, a data e hora de entrada.
    - Se o veículo já saiu, mostra também a data e hora de saída e o valor pago.
    - Se o veículo ainda está estacionado, indica seu status atual.
- **2.6. Sair:**
    - Encerra o programa de forma organizada.

3. Aspectos Técnicos e Implementação

O programa foi desenvolvido atendendo aos seguintes requisitos técnicos e boas práticas:

- **Estrutura do Código:** O código é modularizado e organizado em funções, cada uma com uma responsabilidade específica, melhorando a legibilidade e manutenção.
- **Controle de Fluxo:** Utiliza instruções condicionais (`if-else`, `switch-case`) e laços de repetição (`for`, `do-while`) para controlar o fluxo do programa e responder às interações do usuário.
- **Tipos de Dados e Estruturas:**
    - A estrutura `Veiculo` é utilizada para agrupar as informações de cada veículo (placa, entrada, saída, valor pago, status ativo).
    - Um arranjo (`veiculos[MAX_PLACAS]`) armazena múltiplos objetos `Veiculo`, representando a coleção de veículos no estacionamento.
    - O tipo `time_t` é empregado para armazenar datas e horas como timestamps, facilitando cálculos de tempo de permanência.
- **Manipulação de Ponteiros:** Ponteiros são utilizados implicitamente em operações com strings (arrays de `char`) e explicitamente na passagem de endereços de variáveis para funções como `fscanf` e para retornar múltiplos valores de uma função (ex: `tentarRegistrarSaida`), garantindo manipulação eficiente de dados.
- **Persistência de Dados:** As funções `salvarTodosVeiculos()` e `carregarDados()` são responsáveis por ler e gravar os registros dos veículos em

um arquivo de texto (`registros_completos.txt`). Isso garante que os dados sejam persistidos entre as execuções do programa, não sendo perdidos ao fechar o aplicativo.

- **Interface via Terminal:** Um menu claro é exibido para o usuário, permitindo a navegação entre as diferentes funcionalidades do sistema.
- **Restrição de Interação no `main()`:** Um requisito crucial do projeto foi atendido: **apenas a função `main()` interage diretamente com o usuário para operações de leitura e escrita no terminal**. As demais funções foram refatoradas para receberem os dados necessários como parâmetros e retornarem informações ou um status de sucesso/falha, sem conter chamadas a `printf`, `scanf` ou `fgets` para leitura/escrita de dados de negócio. Funções auxiliares como `lerLinha`, `lerInteiro` e `lerPlaca` foram criadas e são chamadas exclusivamente pelo `main` para encapsular a lógica de entrada segura.
- **Tratamento de Erros e Robustez:**
    - **Validação de Placas:** A função `lerPlaca` foi aprimorada para validar as placas de veículos de acordo com os dois padrões brasileiros (`LLL-NNNN` e `LLLNLNN`). Isso envolve a verificação do comprimento da string, a posição do hífen (se aplicável) e o tipo de caractere (letra ou número) em cada posição, utilizando funções como `isalpha`, `isdigit` e `toupper` para maior precisão e padronização.
    - Validações de entrada do usuário são implementadas para garantir a integridade dos dados (ex: entrada numérica para opções do menu).
    - Abertura de arquivos é verificada para evitar erros de leitura/escrita.
    - Mensagens informativas e de erro são exibidas ao usuário, muitas vezes utilizando cores para melhor destaque.
    - Funções de pausa (`pause()`) são usadas para controlar o fluxo da interface, permitindo que o usuário leia as mensagens antes que a tela seja limpa ou o menu retorne.
    - A utilização de `fgets` combinado com `sscanf` para leitura de entrada do usuário aumenta a robustez, prevenindo "buffer overflows" e falhas de `scanf` com entradas inválidas.
- **Uso de Cores:** Constantes de cores ANSI foram definidas e utilizadas para melhorar a experiência visual do usuário no terminal, destacando informações importantes, mensagens de sucesso, erro e elementos do menu.

4. Conclusão

O Sistema de Gerenciamento de Estacionamento demonstra a aplicação prática de diversos conceitos fundamentais da programação em C, resultando em um programa funcional, organizado e relativamente robusto.