DOCUMENTAÇÃO - SISTEMA BITCAR

Autores: Iago Ribeiro Sales e José Vitor Pires Silveira Campos

Data: 02/12/2025

1. INTRODUÇÃO

1.1 Descrição do Problema

O gerenciamento manual de estoques em revendas de veículos apresenta diversos desafios, como redundância de dados, dificuldade na busca rápida por informações, erros de cálculo em relatórios financeiros e risco de perda de registros físicos. A falta de um sistema centralizado impede a tomada de decisões ágeis e compromete a eficiência operacional do negócio.

1.2 Visão Geral da Solução

O software **BitCar** foi desenvolvido para solucionar esses problemas através da automação do cadastro e controle de veículos. O sistema consiste em uma aplicação de console em linguagem C que permite realizar as operações fundamentais de CRUD (Create, Read, Update, Delete) sobre um banco de dados persistente.

O programa oferece uma interface amigável baseada em menus, controle de acesso via login, relatórios gerenciais exportáveis para planilhas (Excel/CSV) e mecanismos de segurança de dados, como a exclusão lógica, garantindo a integridade das informações.

2. IMPLEMENTAÇÃO

Esta seção detalha a arquitetura do software, as estruturas de dados, as funções principais e as decisões de design adotadas.

2.1 Especificações Técnicas e Bibliotecas

O sistema foi desenvolvido em **Linguagem C** padrão (C99/C11), utilizando compilador GCC (MinGW).

Bibliotecas Utilizadas:

- `<stdio.h>` e `<stdlib.h>`: Manipulação de entrada/saída e alocação de memória.
- `<string.h>`: Manipulação de cadeias de caracteres (ex: `strcmp`, `strcpy`).
- `<conio.h>`: Utilizada para funções de interface de console (`getch`, `getche`) e captura de teclas sem buffer.

- `<windows.h>`: Utilizada para manipulação do cursor (`SetConsoleCursorPosition`) e funções de tempo (`Sleep`).
- `<time.h>`: Captura e formatação da data atual do sistema para registros e relatórios.

2.2 Estrutura de Dados

A base do sistema é a `struct cadastro`, que define o modelo de dados de um veículo. Para otimizar o armazenamento e a legibilidade, foram utilizados `enums` para campos categóricos.

Definição da Estrutura:

```
typedef struct {
   int codigo;             // Chave primária única (auto-incremental)
   char marca[20];         // Marca do veículo
   char modelo[20];        // Modelo do veículo
   int anoFabricacao;      // Ano de fabricação
   int anoModelo;          // Ano do modelo
   Combustivel combustivel; // Enum (1=Gasolina, 2=Alcool, 3=Diesel, 4=Flex)
   TipoCarro tipo;         // Enum (1=Passeio, 2=Utilitario)
   double preco;           // Preço de venda
   char data[12];          // Data da última alteração/inclusão (DD/MM/AAAA)
   int ativo;              // Flag de Exclusão Lógica (1=Ativo, 0=Excluído)
} cadastro;
```

2.3 Arquitetura do Sistema

O sistema segue o paradigma de Programação Estruturada, organizado em uma arquitetura monolítica simplificada, adequada para aplicações de console de pequeno porte. Embora todo o código resida em um único arquivo fonte, as funções foram projetadas para simular uma separação lógica de camadas:

2.3.1 Componentes Lógicos

1. **Camada de Apresentação (Interface):**
    - Composta por funções de desenho de tela (`bordas`, `cabecalho`, `gotoxy`) e interação direta com o usuário (`menu`, `telaLogin`).
    - Isola a lógica visual, permitindo que o usuário interaja com o sistema através de formulários estruturados em vez de linhas de comando puras.
2. **Camada de Regra de Negócios (Lógica):**
    - Funções como `incluir`, `alterar` e `excluir` contêm as regras do sistema (validação de inputs, geração automática de ID, verificação de duplicidade e lógica de exclusão).
    - Responsável por processar os dados brutos inseridos na camada de apresentação antes de enviá-los para armazenamento.

3. **Camada de Persistência (Dados):**
    - Interação direta com o sistema de arquivos do sistema operacional.
    - O sistema não mantém vetores de dados em memória RAM (exceto a struct temporária em uso). Cada operação lê ou escreve diretamente no disco, garantindo persistência imediata.

2.3.2 Fluxo de Dados

O fluxo de informações dentro do sistema segue o seguinte pipeline:

1. **Entrada:** O usuário insere dados via teclado (`stdin`).
2. **Buffer e Validação:** Os dados passam por laços de validação (`do-while`) para garantir consistência (ex: tipos de combustível válidos) e limpeza de buffer.
3. **Memória Volátil:** Os dados validados são armazenados temporariamente em uma `struct cadastro` na memória RAM.
4. **Processamento:** O sistema aplica regras de negócio, como a adição automática da data atual e geração de ID sequencial.
5. **Armazenamento Persistente:** A struct completa é serializada e escrita no arquivo binário `carros.dat` via fluxo de saída.

2.3.3 Decisões de Design

- **Abordagem Sem Vetores Globais:** Optou-se por não carregar todo o banco de dados para a memória RAM no início da execução. Em vez disso, utiliza-se acesso sequencial e aleatório (`fseek`) ao arquivo. Isso permite que o banco de dados cresça indefinidamente sem estourar a memória disponível.
- **Interface TUI (Text User Interface):** O uso da biblioteca `windows.h` para posicionamento de cursor (`gotoxy`) foi escolhido para emular a experiência de sistemas legados profissionais, oferecendo melhor usabilidade e organização visual do que o fluxo padrão de console.

2.4 Persistência de Dados

O sistema utiliza **arquivos binários (`.dat`)** para armazenamento dos dados.

- **Decisão de Design:** Optou-se por arquivos binários em vez de texto (`.txt`) devido à maior eficiência na leitura/escrita de estruturas inteiras (`fread`/`fwrite`) e segurança básica contra edição manual externa.
- **Arquivo Principal:** `carros.dat`

2.5 Descrição das Funcionalidades Principais

2.5.1 Autenticação (`telaLogin`)

Implementa um loop de verificação de credenciais.

- **Segurança:** A senha é mascarada visualmente (exibe * ao digitar) utilizando manipulação de caracteres via `getch()`.
- **Fluxo:** O acesso ao menu principal é bloqueado até que as credenciais corretas ("admin"/"123") sejam inseridas.

2.5.2 Inclusão de Dados (`incluir`)

Responsável por adicionar novos veículos.

- **Geração de ID:** O sistema varre o arquivo para encontrar o maior código existente e soma 1, garantindo unicidade sem necessidade de controle manual.
- **Validação:** Campos como Combustível e Tipo possuem laços de repetição (`do-while`) que impedem a inserção de valores inválidos.
- **Data:** A data de cadastro é gerada automaticamente via `localtime`.

2.5.3 Listagem e Paginação (`listar`, `listarPorCombustivel`)

Exibe os dados em formato tabular desenhado com caracteres ASCII estendidos.

- **Paginação:** Implementada uma lógica que conta as linhas impressas. Ao atingir o limite da tela (linha 22), o sistema pausa e limpa a tela antes de continuar, evitando rolagem descontrolada.
- **Filtro:** A função `listarPorCombustivel` aplica uma condição `if` dentro do laço de leitura, exibindo apenas registros que coincidem com o critério do usuário.

2.5.4 Alteração e Exclusão (`alterar`, `excluir`)

Estas funções manipulam dados existentes.

- **Busca Prévia:** Utilizam a função auxiliar `acharCodigo` para localizar e exibir o registro antes de qualquer modificação.
- **Manipulação de Arquivo:** Utilizam o modo de abertura `"rb+"` (leitura e atualização).
- **Posicionamento:** Uso da função `fseek(arq, -(long)sizeof(cadastro), SEEK_CUR)` para retroceder o cursor do arquivo e sobrescrever o registro exato.

2.5.5 Exclusão Lógica (Decisão de Arquitetura)

Foi adotada a estratégia de **Exclusão Lógica** em detrimento da Exclusão Física.

- **Funcionamento:** Ao "excluir" um carro, o campo `ativo` é alterado para `0`. O registro permanece no arquivo, mas é ignorado nas listagens e buscas.
- **Justificativa:** Esta abordagem é mais segura (permite futura implementação de restauração/lixeira), mais rápida (evita reescrever todo o arquivo temporário) e mantém o histórico de dados.

2.5.6 Relatórios Avançados (`gerarRelatorio`)

Gera um arquivo externo compatível com Excel (`.csv`).

- **Dashboard:** Além de exportar a lista, a função calcula estatísticas em tempo real (Total em Estoque, Média de Preço, Carro Mais Caro/Barato) e as anexa ao final do arquivo.
- **Tratamento de Erros:** Verifica se o arquivo de destino está aberto em outro programa antes de tentar gravar, evitando travamento (crash) do sistema.

3. CONCLUSÃO

O desenvolvimento do projeto BitCar permitiu a aplicação prática de conceitos avançados de programação estruturada, manipulação de arquivos e lógica de sistemas comerciais. O software atende a todos os requisitos funcionais propostos, oferecendo uma solução robusta para o gerenciamento de veículos.

3.1 Dificuldades Encontradas e Soluções

Durante a implementação, as principais dificuldades foram:

1. **Manipulação do Buffer de Teclado:** A mistura de `scanf` (para números) e `fgets` (para strings) causava "pulos" nos formulários devido à permanência do caractere `\n` no buffer.
    - *Solução:* Implementação de um laço de limpeza de buffer `while ((c = getchar()) != '\n' && c != EOF);` após leituras numéricas, substituindo o uso não-padrão de `fflush(stdin)`.
2. **Manipulação do Cursor de Arquivo:** Entender o posicionamento correto do ponteiro do arquivo após uma leitura (`fread`) para realizar a sobrescrita (`fwrite`) correta.
    - *Solução:* Estudo da função `fseek` e uso de *casts* para `long` para evitar warnings de compilador.
3. **Visualização em Tabela:** Alinhar colunas no console com dados de tamanhos variáveis.
    - *Solução:* Uso de especificadores de formato do `printf` (ex: `%.15s` para limitar strings).

O resultado final é um sistema estável, com tratamento de erros eficiente e uma interface visualmente agradável.

4. BIBLIOGRAFIA

DevDocs. Disponível em: <https://devdocs.io/c/>. Acesso em: 2 dez. 2025.

MEO, Code. Sistema com arquivo binário - parte 1. Disponível em: <https://www.youtube.com/watch?v=E02sjMnI8_w&list=PLsNXjPIh8RkURr_DfDJhg1sw2S1FY8loH>. Acesso em: 2 dez. 2025.