**Documentação do Projeto: Sistema de Controle de Estoque em C**

Este documento descreve a estrutura e a organização de um sistema de controle de estoque desenvolvido em linguagem C, focado em modularidade, clareza e aderência a requisitos de programação fundamentais.

**Descrição do Sistema**

O sistema de controle de estoque permite o gerenciamento de produtos através de operações de entrada e saída. Ele gerencia o cadastro de produtos com código, nome, quantidade e preço. A cada movimentação (entrada ou saída), o estoque é atualizado e o histórico é salvo em arquivo.

**Requisitos Mínimos do Projeto**

O desenvolvimento do sistema segue os seguintes requisitos:

* **REQ1: Possuir uma matriz, um vetor ou struct.**
* **REQ2: Ler dados de entrada do usuário.**
* **REQ3: Organizado em mais de uma função (além da função main).**
* **REQ4: Ao menos um parâmetro passado por referência.**
* **REQ5: Usar o operador de desreferência.**
* **REQ6: Salvar o resultado em um arquivo de texto.**

**Estrutura de Pastas do Projeto**

A organização do projeto em diretórios e arquivos segue um padrão modular para facilitar a manutenção e o desenvolvimento:

controle\_estoque/

├── src/

│ ├── main.c # Contém apenas a função main e o loop principal do menu.

│ ├── produto\_crud.c # Implementações das funções de cadastro, listagem, atualização.

│ ├── validacoes.c # Implementações de todas as funções de validação de entrada.

│ ├── arquivo\_io.c # Implementações das funções de leitura/escrita em arquivo.

│ └── utilidades.c # Implementações de funções genéricas (limpar tela, buffer, etc.).

├── include/

│ ├── produto.h # Definição da struct Produto e protótipos das funções de CRUD.

│ ├── validacoes.h # Protótipos das funções de validação.

│ ├── arquivo\_io.h # Protótipos das funções de manipulação de arquivos.

│ └── utilidades.h # Protótipos das funções utilitárias.

├── data/

│ ├── estoque.csv # Arquivo para salvar o estado atual do estoque.

│ └── movimentacoes.csv # Arquivo para salvar o histórico de entradas e saídas.

└── Makefile # Arquivo para compilar o projeto.

**Explicação da Organização e Funções Pequenas**

A chave para um código organizado e profissional, com funções pequenas, é o princípio da **"Responsabilidade Única"**: cada função deve fazer **apenas uma coisa**, e fazê-la bem. Isso torna o código mais claro, modular, reutilizável e fácil de depurar e manter.

**Detalhamento dos Módulos e Funções:**

**1. Módulo main.c**

* **Responsabilidade:** Orquestrar o fluxo principal do programa.
* **Conteúdo:** Contém apenas a função main().
* **Funções Chamadas:** limparTela(), printf() para o menu, scanf() para a opção do menu, limparBuffer(), e chamadas para as funções de alto nível dos outros módulos (ex: cadastrarProduto(), atualizarEstoque(), listarProdutos(), salvarEstoque()).
* **REQs Atendidos:**
  + **REQ2 (Ler dados de entrada do usuário):** Indiretamente, através da leitura da opção do menu.
  + **REQ3 (Organizado em mais de uma função):** A main é uma função que chama outras.

**2. Módulo include/produto.h e src/produto\_crud.c**

* **Responsabilidade:** Definir a estrutura Produto e implementar as operações de CRUD (Create, Read, Update) e busca de produtos.
* **include/produto.h:**
  + **REQ1 (Struct):** Contém a definição da struct Produto.
  + Protótipos das funções de CRUD (cadastrarProduto, listarProdutos, atualizarEstoque).
* **src/produto\_crud.c:**
  + **cadastrarProduto(Produto produtos[], int \*totalProdutos):**
    - **Responsabilidade:** Gerenciar o fluxo de cadastro de um novo produto.
    - **REQ4 (Parâmetro por referência):** Recebe totalProdutos por referência para atualizar o contador global de produtos.
    - **REQ5 (Operador de desreferência):** Usará \*totalProdutos para acessar e modificar o valor.
    - **Funções Chamadas (Exemplos de Funções Pequenas):**
      * lerNomeProduto(char \*nome\_buffer, int buffer\_size): Apenas lê o nome do produto.
      * validarNomeDuplicado(Produto produtos[], int total, const char \*nome): Verifica se o nome já existe (retorna 1 ou 0).
      * lerQuantidadeProduto(): Apenas lê a quantidade inicial.
      * validarQuantidade(int qtd): Valida se a quantidade é não negativa.
      * lerPrecoProduto(): Apenas lê o preço.
      * validarPreco(float preco): Valida se o preço é não negativo.
      * registrarMovimentacao(): Para logar o cadastro.
      * salvarEstoque(): Para persistir o estoque atual.
      * limparTela(), pausarTela().
  + **listarProdutos(Produto produtos[], int totalProdutos):**
    - **Responsabilidade:** Exibir todos os produtos cadastrados em formato de tabela.
    - **Funções Chamadas:** limparTela(), printf() para cabeçalhos e linhas da tabela, pausarTela().
  + **atualizarEstoque(Produto produtos[], int totalProdutos, int tipoMovimentacao):**
    - **Responsabilidade:** Gerenciar o fluxo de entrada ou saída de produtos.
    - **Funções Chamadas:**
      * lerTermoBusca(char \*buffer, int buffer\_size): Lê o código ou nome do produto para buscar.
      * buscarProdutoPorCodigoOuNome(Produto produtos[], int total, const char \*termo): Retorna o índice do produto ou -1.
      * lerQuantidadeMovimentacao(): Lê a quantidade a ser movimentada.
      * validarQuantidade(int qtd): Valida se a quantidade é positiva.
      * registrarMovimentacao(): Para logar a movimentação.
      * salvarEstoque(): Para persistir o estoque atual.
      * limparTela(), pausarTela().

**3. Módulo include/validacoes.h e src/validacoes.c**

* **Responsabilidade:** Implementar todas as funções de validação de entrada de dados e de lógica de negócio.
* **Conteúdo:**
  + validarQuantidade(int qtd): Retorna 1 se qtd >= 0, 0 caso contrário.
  + validarPreco(float preco): Retorna 1 se preco >= 0.0, 0 caso contrário.
  + validarOpcaoMenu(int opcao, int min, int max): Retorna 1 se opcao está entre min e max, 0 caso contrário.
  + validarNomeDuplicado(Produto produtos[], int total, const char \*nome): Retorna 1 se nome já existe no array produtos, 0 caso contrário.
  + **REQ2 (Ler dados de entrada do usuário):** Aqui estarão as funções que realizam a leitura e validação em um só passo, garantindo que o dado retornado já seja válido.
    - lerInteiroComValidacao(const char \*mensagem, int min, int max): Lê um inteiro, valida e repete até ser válido.
    - lerFloatComValidacao(const char \*mensagem, float min, float max): Lê um float, valida e repete até ser válido.
    - lerStringComValidacao(const char \*mensagem, char \*buffer, int buffer\_size): Lê uma string, remove \n, e valida (ex: não vazia).

**4. Módulo include/arquivo\_io.h e src/arquivo\_io.c**

* **Responsabilidade:** Gerenciar a leitura e escrita de dados nos arquivos de persistência (estoque.csv e movimentacoes.csv).
* **REQ6 (Salvar o resultado em um arquivo de texto):** Todas as operações de arquivo estarão aqui.
* **Conteúdo:**
  + carregarEstoque(Produto produtos[], int \*totalProdutos): Lê data/estoque.csv para popular o array produtos no início do programa.
  + salvarEstoque(Produto produtos[], int totalProdutos): Sobrescreve data/estoque.csv com o estado atual do estoque.
  + registrarMovimentacao(const Produto\* produto, int quantidade, int tipo): Adiciona uma nova linha de log a data/movimentacoes.csv.

**5. Módulo include/utilidades.h e src/utilidades.c**

* **Responsabilidade:** Fornecer funções genéricas e de baixo nível que são úteis em várias partes do programa, mas não estão diretamente ligadas à lógica de negócio do estoque.
* **Conteúdo:**
  + limparBuffer(): Limpa o buffer de entrada (stdin).
  + limparTela(): Limpa o console (cls ou clear).
  + pausarTela(): Pausa a execução e espera por uma tecla/Enter.
  + strcasecmp\_custom(const char \*s1, const char \*s2): Versão portátil de strcasecmp para comparação de strings sem diferenciar maiúsculas/minúsculas.

Com essa estrutura, cada arquivo .c implementa as funções que foram prototipadas em seu respectivo arquivo .h, e a main orquestra tudo chamando as funções de alto nível.