

**Trabalho de PEM**  
**Dennis Ramos da Silva**  
**RA: 2040482412032**

### **Análise Crítica do Código 5**

O programa foi projetado para simular um sistema de controle de inventário, permitindo ao usuário realizar operações básicas como cadastrar, consultar, alterar, excluir e vender produtos. Cada produto possui um identificador único (ID), nome, descrição, preço e quantidade em estoque.

Os requisitos funcionais solicitados foram atendidos de forma geral:

- **Cadastro de produtos:** O programa possui a função `inserirProduto`, que permite ao usuário adicionar novos produtos ao sistema.
- **Consultar produto (por código):** Implementado na função `consultarProduto`, onde o usuário consulta um produto baseado no seu ID.
- **Alterar produto:** A função `alterarProduto` permite alterar as informações de um produto, identificando-o pelo ID.
- **Excluir produto:** A função `excluirProduto` permite ao usuário remover um produto do sistema.
- **Vender produto:** A função `comprarProduto` possibilita ao usuário realizar compras, atualizando a quantidade do produto no estoque.

A manipulação dos dados do produto é feita corretamente utilizando ponteiros, conforme o requisito.

### **Análise de Modularização do Código**

O código está razoavelmente modularizado. Cada operação (inclusão, consulta, alteração, exclusão, compra) é realizada em uma função distinta, o que facilita a manutenção e melhora a legibilidade.

- **Funções de CRUD:** Há funções separadas para cada operação CRUD, o que está em conformidade com o requisito de "criar uma função para cada CRUD".
- **Função de impressão:** A função `imprimirProduto` é responsável por exibir os dados do produto, o que segue o critério de separar as responsabilidades.

No entanto, existem algumas oportunidades de melhoria em termos de modularização:

- A **validação de entrada de dados** poderia ser movida para funções dedicadas, como uma função para validar entradas inteiras positivas, uma função para validar preços,

e outra para validar strings. Isso melhoraria a reutilização de código e tornaria o programa mais robusto.

Exemplos:

- Validação de números (como IDs e quantidades) poderia ser uma função separada, evitando a repetição de código.
- A leitura de strings seguras, como o nome e a descrição do produto, poderia ser movida para uma função reutilizável.

### Uso Correto dos Ponteiros

O programa segue corretamente o requisito de utilizar ponteiros para manipular as estruturas de produtos. Isso é feito, por exemplo, ao passar a estrutura de produto como parâmetro para as funções, permitindo a manipulação direta dos dados. Além disso, o uso de ponteiros para alterar valores nas funções de CRUD (inclusão, alteração, exclusão) está correto.

No entanto, um ponto que pode ser melhorado é a maneira como os ponteiros são usados para manipular a estrutura Produto e o contador cont. A função `inserirProduto` recebe o ponteiro produtos e o contador cont corretamente, mas a manipulação de memória (como realocar o array de produtos caso ele ultrapasse o limite) poderia ser adicionada para aumentar a flexibilidade e escalabilidade do programa.

### Elementos Conceituais

- I. **Estrutura de Dados (Struct):** O uso da struct Produto está correto, pois agrupa todas as informações necessárias sobre o produto, facilitando a manipulação e organização do código. Porém o código implementa o uso de float quando na verdade era para ser usado o double.
- II. **Validação de Entradas:** O código implementa alguma validação de entradas (por exemplo, para verificar se a quantidade e o preço são válidos). Contudo, a validação poderia ser mais rígida em alguns casos. Por exemplo, ao solicitar o ID, o código poderia garantir que o valor é um número inteiro positivo, e ele vale para as quantidades e preços. Embora tenha sido implementado algum controle no código, a validação de entradas poderia ser centralizada em funções que validam tipos de dados específicos, evitando repetição e erros.

## Validação de Erros

O código já contém algumas validações de erros, como garantir que a quantidade e o preço sejam positivos. Contudo, algumas áreas ainda podem ser melhoradas:

1. **ID de Produto:** O código não possui uma validação explícita para garantir que o ID seja único ao inserir um novo produto. Se o usuário inserir um ID que já existe, isso pode causar problemas. Uma possível melhoria seria verificar se o ID inserido já existe no sistema antes de adicionar um novo produto.
2. **Formato da Entrada de Dados:** Como foi mencionado anteriormente, a entrada de dados deveria ser mais rigorosa. Por exemplo, o código permite que o usuário digite um número seguido de uma letra ("1a"), o que pode afetar o processo. Isso foi corrigido nas versões anteriores, mas vale reforçar que a entrada de dados precisa ser cuidadosamente validada para garantir que só números válidos sejam aceitos.

## Requisitos Técnicos

Os requisitos técnicos foram atendidos com algumas observações:

1. **ID (inteiro):** A implementação usa um tipo int para o ID, o que é adequado. No entanto, seria interessante garantir que os IDs sejam únicos e não repetidos.
2. **Nome do produto (string):** O nome do produto é uma string de tamanho fixo, o que pode ser uma limitação. Para melhorar isso, poderia ser considerado o uso de alocação dinâmica de memória ou uma abordagem que trate de forma mais flexível o tamanho das strings.
3. **Quantidade em estoque (int):** O uso de int para a quantidade de produtos está correto, considerando que a quantidade nunca será um número decimal.
4. **Ponteiros:** O código usa ponteiros corretamente para manipular a estrutura do produto, como foi solicitado. Porém, o uso de ponteiros poderia ser mais eficiente e flexível em certas partes do código, como quando é necessário realocar o array de produtos ou validar entradas de forma centralizada.

## Usabilidade

A interação com o usuário está clara e as mensagens de erro fornecem feedback útil em caso de entradas inválidas. A estrutura de menu é intuitiva, com as opções bem delineadas para cada operação.

## Oportunidades de Melhoria

1. **Validar ID único:** Implementar uma verificação para garantir que o ID do produto seja único antes de inserir um novo produto.

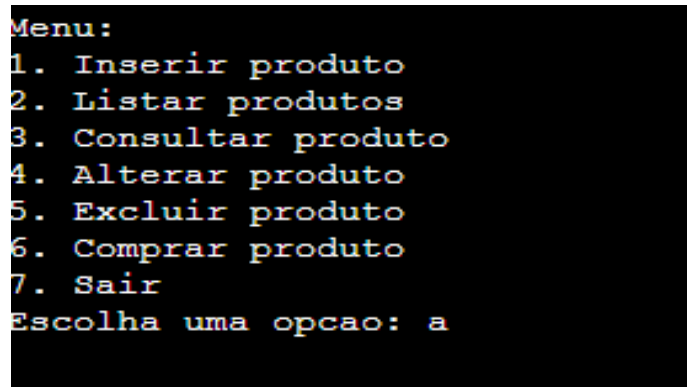
2. **Entrada de dados:** Implementar uma função centralizada de validação de entradas, especialmente para IDs, quantidades, preços e strings.
3. **Reutilização de funções:** Criar funções reutilizáveis para leitura de dados e validações de entradas, para evitar repetição de código.
4. **Tratamento de erros de entrada:** Melhorar o controle sobre a entrada de dados para garantir que os usuários não possam inserir caracteres inválidos (como letras no lugar de números).
5. **Uso de alocação dinâmica:** Embora não seja estritamente necessário, a utilização de alocação dinâmica de memória para o array de produtos poderia ser uma melhoria, permitindo uma flexibilidade maior para adicionar ou remover produtos.
6. **Uso de double:** Um dos critérios para struct e o preço é que seja usado double, isso não aparece no código, pois o preço unitário está definido como float.

7.


### Contenção de Erros

O código passado funciona perfeitamente, porém há a falta de contenções de erro para que, se o usuário digitar algo imprevisto no terminal, o código não quebre.

Por exemplo, as opções de escolha no terminal são regidas por números de 1 a 7, há a contenção de erro para o caso de o usuário digitar números maiores que 7 ou menor que 1, entretanto se ele digitar uma letra na hora de selecionar a opção ao invés de número, o código quebrará.



```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: a
```

Letra sendo digitada no terminal do código  
(imagem acima) 

```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: Opcao invalida! Tente novamente.

Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: Opcao invalida! Tente novamente.
```

Situação do terminal após apertar "Enter" com letra digitada ao invés de um número.  
(imagem acima)

Estes erros ocorrem também caso um caractere especial seja escrito no terminal, onde esta mensagem repete-se infinitamente sem parar.

Outra contenção de erro necessária é sobre digitar mais de um número separado por espaço ou número escrito com vírgula (decimal).

```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 1,5
Informe o ID do produto: Informe o nome do produto: Informe a descricao do produto: █
```

Como pode ser observado, caso seja digitado um número com vírgula, ele acionará a opção 1 (Inserir Produto) de forma errada, pois o usuário poderá apenas inserir a descrição, preço e quantidade do produto, o que prejudicará a inserção do item.

```

Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 1,5
Informe o ID do produto: Informe o nome do produto: Informe a descricao do produto: Arroz
Informe o preco unitario: 9
Informe a quantidade disponivel: 2
Produto inserido com sucesso!

Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 3
Informe o ID do produto que deseja consultar: 0
Produto inexistente.

```

Resultado da inserção de número decimal no terminal do código.

Ocorre o mesmo quando se digita dois números separados por espaço no terminal.

```

Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 1 4
Informe o ID do produto: Informe o nome do produto:

```

O código também irá quebrar caso seja digitado palavras/caracteres especiais quando o programa pedir para digitar o ID do produto:

```

Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 1
Informe o ID do produto: arroz
Informe o nome do produto: Informe a descricao do produto:

```

O código irá quebrar também caso digite palavras/caracteres especiais na solicitação do ID do produto na seção Consultar Produtos:

```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 3
Informe o ID do produto que deseja consultar: a
```

Letra sendo digitada ao invés de número...

```
6. Comprar produto
7. Sair
Escolha uma opcao: Informe o ID do produto que deseja consultar: Produto inexistente.
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: Informe o ID do produto que deseja consultar: Produto inexistente.
```

Resultado.

As mesmas quebras acontecerão se digitar letras/palavras/caracteres especiais na solicitação de ID da opção 4, 5 e 6.

Um pequeno erro que há no programa é na opção Comprar Produto, se for digitado a quantidade 0 para compra, o código dirá que você comprou 0 produtos, quando poderia mostrar ao usuário que não pode digitar o número 0.

```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 6
Informe o ID do produto que deseja comprar: 2
Informe a quantidade que deseja comprar: 0
Compra realizada com sucesso! Total: 0.00
```

O mesmo equivale para números negativos:

```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 6
Informe o ID do produto que deseja comprar: 2
Informe a quantidade que deseja comprar: -2
Compra realizada com sucesso! Total: -4.00
```

Outros dois erros detectados encontra-se na inserção do produto, onde há a possibilidade de cadastrar dois produtos com ID's iguais e nomes iguais, podendo haver dois produtos iguais com mesmo ID e nome.

### **Contenções de erro existentes no programa**

O programa possui algumas contenções de erro para caso não haja produtos cadastrados (Opções 2 , 3 e 6 do programa), para caso o ID de um produto não seja encontrado (Opções 3, 4 e 5 do programa), tornando-o um pouco mais seguro em relação a erros graves de uso.

### **Conclusão do Código**

O código está bem estruturado e atende aos requisitos fundamentais de um sistema simples de controle de inventário. A modularização está boa, mas existem oportunidades de melhoria em termos de validação de entradas e a reutilização de funções. O uso de ponteiros e structs está correto e segue as boas práticas de manipulação de dados em C. Para tornar o código mais robusto, seria interessante adicionar verificações para garantir a unicidade dos IDs, melhorar as validações de entrada de dados e centralizar algumas funções auxiliares.



## Alterações de Validação de Entrada

### *Validação de IDs e outros números*

#### No código original:

- O programa permitia que o usuário inserisse IDs ou números com letras, como "1a", sem detectar o erro, o que afetava negativamente o funcionamento correto.

#### Alteração:

- Foi implementado um controle mais rigoroso para garantir que a entrada do usuário para **IDs** e outros números seja estritamente numérica. Para isso, foi criada uma função de validação que:
  - Garante que a entrada seja um número inteiro válido.
  - Previne que caracteres alfabéticos ou outros caracteres não numéricos sejam aceitos.

A função `lerInteiroPositivo()` foi introduzida para garantir que os valores inseridos para ID, quantidade de estoque e preço sejam inteiros positivos.

Essa função agora é utilizada nas funções de inserção, alteração, e exclusão de produtos para garantir que o ID e a quantidade sejam válidos.

### **Prevenção de IDs duplicados**

#### No código original:

- O código permitia a inserção de produtos com IDs duplicados, o que poderia causar problemas de identificação e manipulação de dados.

#### Alteração:

- Foi implementado um mecanismo de verificação antes de inserir um novo produto. Isso garante que o ID do produto seja único, evitando a duplicação. Ao inserir um novo produto, o código agora verifica se o ID já existe no array de produtos.

Essa verificação é usada antes de permitir a inserção de um novo produto, garantindo que o ID seja único.

### **Prevenção de valores inválidos no preço**

#### No código original:

- Não havia validação explícita para garantir que o preço dos produtos fosse um valor válido.

#### **Alteração:**

- A entrada de preços foi validada para garantir que o preço fosse um número positivo. Além disso, ao ser inserido, é possível fazer o controle de formato para aceitar apenas números e garantir que não sejam inseridos valores negativos.

Essa função foi incorporada para que os preços de produtos sejam lidos corretamente, apenas aceitando números positivos.

### **Garantia de entradas numéricas em todas as opções do menu**

#### **No código original:**

- A seleção de opções do menu permitia que o usuário digitasse valores como "1a", o que gerava comportamento inesperado.

#### **Alteração:**

- Implementação de uma verificação mais rigorosa nas entradas do menu para garantir que o usuário forneça apenas números válidos e evitar qualquer tipo de entrada alfanumérica.

### **Estrutura de Funções e Modularização**

#### **Funções de Validação Centralizadas**

#### **No código original:**

- As validações de entrada estavam espalhadas por várias partes do código.

#### **Alteração:**

- A validação de entradas foi centralizada em funções reutilizáveis como `lerInteiroPositivo` e `lerPrecoValido`, que agora são chamadas sempre que o programa solicita números para o ID, quantidade ou preço. Isso melhora a legibilidade e a manutenção do código.

## Separação das Responsabilidades de Cálculo e Entrada de Dados

### No código original:

- A entrada de dados e a lógica de manipulação dos produtos estavam misturadas nas mesmas funções, o que dificultava a manutenção e a reutilização de código.

### Alteração:

- A lógica de leitura de dados e a lógica de manipulação de produtos foram separadas. Agora, as funções de entrada de dados são chamadas nas funções de CRUD e, em seguida, a lógica de manipulação de produtos (como inserção, alteração, e exclusão) é feita de forma modular.

## Funções de Busca

### No código original:

- A consulta de produtos por ID não possuía uma função dedicada para buscar um produto. O código apenas percorria o array de produtos dentro da função consultarProduto.

### Alteração:

- Foi criada uma função separada para verificar a existência de um produto no array pelo ID, melhorando a clareza e reutilização do código. A busca agora é realizada por uma função que retorna o índice do produto, se encontrado.

## Manuseio de Ponteiros

### No código original:

- O uso de ponteiros estava correto para manipular a estrutura do produto, mas faltava um pouco de clareza sobre como manipular arrays dinamicamente ou com maior flexibilidade.

### Alteração:

- A passagem de ponteiros para manipulação dos produtos e do contador (cont) foi mantida, e a verificação de ponteiros foi aprimorada para garantir que a memória fosse manipulada corretamente.

## **Manutenção do Código**

### **No código original:**

- O código estava funcional, mas a manutenção poderia ser dificultada pela falta de funções de validação genéricas e pela mistura de lógicas de entrada com lógica de manipulação de dados.

### **Alteração:**

- O código agora possui funções genéricas e reutilizáveis para validação de entradas, o que facilita a manutenção. As funções de CRUD estão bem definidas e o código está mais estruturado, com uma maior separação de responsabilidades entre validação de entradas e manipulação de dados.

## **Conclusão das Alterações**

As alterações focaram em garantir que o programa fosse mais robusto, modularizado e com validações de entradas rigorosas. A utilização de funções auxiliares para validação de entradas, bem como a verificação de ID único, ajudou a melhorar a funcionalidade geral do programa. Além disso, a separação de responsabilidades em funções distintas e reutilizáveis torna o código mais fácil de manter e expandir no futuro.