Análise crítica de codigo - PEM

Pedro Yun Han RA: 2040482412033

1. Modularização

O código está estruturado em várias funções, cada uma responsável por uma tarefa específica, o que promove a modularização e facilita a manutenção e extensão do programa. A modularização é uma prática essencial em desenvolvimento de software, pois divide o código em partes menores e mais gerenciáveis. As funções principais incluem:

- **menuOpcoes**: Exibe o menu de opções ao usuário e retorna a escolha.
- **incluirProduto**: Registra um novo produto no sistema.
- **alterarProduto**: Permite a alteração dos dados de um produto existente.
- **excluirProduto**: Remove um produto do sistema.
- **comprarProduto**: Registra a compra de um produto, atualizando a quantidade em estoque.
- **consultarProduto**: Consulta e exibe os dados de um produto específico.
- **listarProdutos**: Lista todos os produtos cadastrados no sistema.
- **mudarProdutos**: Auxilia na reordenação dos produtos.
- **checarCodigoValido**: Verifica se um código de produto é válido.
- **confirmarResposta**: Solicita confirmação do usuário para ações críticas.

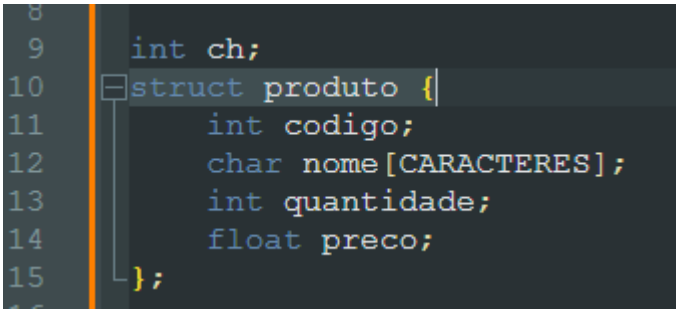
Exemplos de Código

```c
int menuOpcoes() {
    int resposta;
    printf("\nO que deseja fazer?\n");
    printf("0 - Finalizar compra\n");
    printf("1 - Incluir um produto\n");
    printf("2 - Alterar um produto\n");
    printf("3 - Excluir um produto\n");
    printf("4 - Comprar um produto\n");
    printf("5 - Consultar um produto\n");
    printf("6 - Ver lista de produtos\n");
    scanf("%i",&resposta);
    while((ch = getchar()) != '\n' && ch != EOF);
    if(resposta < 0 || resposta > 6) {
        printf("Resposta inválida!\n");
        printf("Digite o numero ao lado da ação que deseja tomar\n");
        menuOpcoes();
    }
    return resposta;
}
```

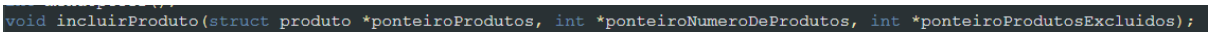
2. Elementos Conceituais

O código utiliza conceitos fundamentais da linguagem C, incluindo:

- **Structs**: Utilizada para definir a estrutura de dados de um produto, contendo o ID, nome, quantidade em estoque e preço.



- **Ponteiros**: Utilizados para manipular as estruturas, facilitando a passagem de parâmetros por referência e economizando memória.



- **Funções**: Cada operação principal é implementada como uma função separada, promovendo a modularização e reutilização do código.

3. Elementos de Negócio (Requisitos)

Os elementos de negócio especificados foram atendidos da seguinte forma:

- **Cadastro de Produtos**: Implementado na função `incluirProduto`.

- **Alteração de Produtos**: Implementado na função `alterarProduto`.

- **Exclusão de Produtos**: Implementado na função `excluirProduto`.

- **Venda de Produtos**: Implementado na função `comprarProduto`.

- **Consulta de Produtos**: Implementado na função `consultarProduto`.

- **Listagem de Produtos**: Implementado na função `listarProdutos`.

4. Regras Explícitas

- **Limitação de Cadastro**: O cadastro está limitado a 500 produtos, conforme definido pela constante `MAXPRODUTOS`.

```
#define MAXPRODUTOS 500
```

- **Validação de Código de Produto**: A função `checarCodigoValido` verifica a existência do produto antes de realizar operações.

- **Operações CRUD**: Funções específicas foram criadas para incluir, alterar, excluir e consultar produtos, utilizando ponteiros para manipular a estrutura do produto.

5. Regras Implícitas

- **Fluxo de Controle**: O código retorna ao menu principal após a conclusão de cada operação, garantindo uma boa experiência de usuário.
- **Limpeza do Buffer de Entrada**: Após a utilização de `scanf`, o buffer de entrada é limpo para evitar problemas com entradas subsequentes.

```
    while((ch = getchar()) != '\n' && ch != EOF);
```

- **Confirmação de Operações**: As operações críticas, como exclusão e compra de produtos, exigem confirmação do usuário, implementado pela função `confirmarResposta`.

```
int confirmarResposta() {
    int resposta;
    printf("0 - Não/ 1 - Sim\n");
    scanf("%i",&resposta);
    return resposta;
}
```

6. Implementação das Funções de CRUD

Incluir Produto

A função `incluirProduto` é responsável por adicionar um novo produto ao sistema. Esta função solicita ao usuário as informações necessárias, como nome, preço e quantidade, e armazena esses dados na estrutura do produto.

Alterar Produto

A função `alterarProduto` permite ao usuário modificar os dados de um produto existente. Isso inclui a alteração do nome, preço e quantidade em estoque.

Excluir Produto

A função `excluirProduto` remove um produto do sistema. Ela solicita ao usuário o código do produto a ser excluído e, após confirmação, realiza a exclusão.

Consultar Produto

A função `consultarProduto` permite ao usuário consultar os dados de um produto específico com base no seu código.

Listar Produtos

A função `listarProdutos` exibe todos os produtos cadastrados no sistema em uma tabela formatada.

7. Boas Práticas e Considerações Finais

Modularização e Reutilização

A modularização do código facilita a reutilização de funções e a manutenção do sistema. Cada função está claramente definida e realiza uma única tarefa, o que melhora a clareza e a organização do código.

Uso de Ponteiros

O uso de ponteiros é essencial para a manipulação eficiente das estruturas de dados. Isso permite operações dinâmicas e economiza memória, pois os dados são manipulados diretamente.

Validação de Entrada

A validação de entrada é um aspecto crucial que garante a integridade dos dados. A função `checarCodigoValido` é um exemplo de como a validação pode ser implementada para verificar a existência de produtos antes de realizar operações.

Confirmação de Ações

A confirmação de ações críticas, como a exclusão e compra de produtos, protege contra operações acidentais. Isso é implementado pela função `confirmarResposta`, que solicita confirmação do usuário antes de prosseguir.

Limpeza do Buffer de Entrada

A limpeza do buffer de entrada após o uso de `scanf` evita problemas com entradas subsequentes, garantindo que os dados sejam lidos corretamente.

Conclusão

O código analisado atende aos requisitos especificados de forma modular e eficiente. A utilização de `structs`, ponteiros e funções promove a clareza e manutenção do código. As regras explícitas e implícitas garantem a integridade dos dados e a correta operação do sistema. Porém há espaço para melhorias. Quando opções inválidas são imputadas o código pode quebrar.

Melhorias

1. **menuOpcoes**: Corrigir para retornar chamada recursiva em caso de resposta inválida.
2. **incluirProduto**: Adicionar validação para nome, preço e quantidade.
3. **alterarProduto**: Adicionar validação para código, nome, preço e quantidade.
4. **excluirProduto**: Adicionar validação para código.
5. **comprarProduto**: Adicionar validação para código e quantidade.
6. **consultarProduto**: Adicionar validação para código.
7. **listarProdutos**: Não há alteração necessária
8. **mudarProdutos**: Melhorar a lógica de troca de produtos.
9. **checarCodigoValido**: Não há alteração necessária
10. **confirmarResposta**: Adicionar validação para resposta.
11. **main**: Inicialização correta das variáveis e funcionamento esperado do loop principal.

Com essas correções, o código deve funcionar corretamente, evitando que números sejam aceitos em lugares não apropriados e garantindo que o código não quebre.