

Trabalho de PEM

Análise crítica do código

Cauê Ferreira Lacerda RA: 2040482412015

Primeiras impressões do código

O código não tem nenhum erro de sintaxe que impeça o mesmo de rodar, aparenta estar bem modularizado, usa camelCase e utiliza os requisitos (ponteiro, funções, struct, e valores de variáveis parcialmente correto), pois não utiliza o valor “double” para a variável que armazena o valor do produto.

Análise do código recebido

➤ Pontos positivos

A estrutura (Struct) foi utilizada para cadastrar o produto conforme a requisição da empresa. Além disso, o “struct” já contém as informações exigidas pela empresa (ID, nome do produto, quantidade em estoque e valor do produto).

```
typedef struct {
    int id;
    char nome[NOME_MAX];
    char descricao[DESCRICAO_MAX];
    float precoUnitario; // Usar float para valores monetários
    int qteDispo;
} Produto;
```

O código segue “boas maneiras”, por exemplo camelCase para o nome das variáveis, utiliza variáveis globais para o tamanho máximo de produtos e para o tamanho máximo do nome e da descrição, algo essencial quando se trabalha em um projeto grande, pois facilita na troca de valor/tamanho do seu array e afins te poupando tempo. Outro ponto positivo é a nomeação das variáveis/funções, sendo claro e direto.

```
#define MAX_PRODUTOS 100
#define NOME_MAX 50
#define DESCRICAO_MAX 100
```

```
void inserirProduto(Produto *produtos, int *cont);
void consultarProduto(Produto *produtos, int cont);
void alterarProduto(Produto *produtos, int cont);
Void excluirProduto(Produto *produtos, int *cont);
```

Além disso, ele também segue os requisitos da empresa em relação as funções, pois ele faz uma função para cada elemento do CRUD (criar, ler, atualizar, deletar) e utiliza-se de ponteiro para se comunicar entre as funções e para receber/manipular os elementos armazenados na memória. Além de tudo usa ponteiro para imprimir os dados.

```
void imprimirProduto(Produto *produto) {
    printf("ID: %d, Nome: %s, Descricao: %s, Preco: %.2f, Estoque: %d\n",
           produto->id, produto->nome, produto->descricao, produto->precoUnitario, produto->qteDispo);
}

void consultarProduto(Produto *produtos, int cont) {
    int idProduto;
    printf("Informe o ID do produto que deseja consultar: ");
    scanf("%d", &idProduto);
```

Aliás, o programa apresenta um menu bem interativo com opções enumeradas, facilitando a visualização das opções para o usuário e tem as mensagens que ilustram se a ação foi bem sucedida ou não. Além do mais que o código permite a leitura da “string” com espaço (scanf(“ %[^\n]”,)).

Menu

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

➤ Pontos negativos

Primeiro ponto negativo é que a variável “precoUnitario” não utiliza “double” como pedido pela empresa, mas sim “float”. Fora isso o código não tem a opção “aplicar desconto” que também foi requisitado pela empresa.

Segundo ponto negativo é a falta de verificação/contenção de algumas coisas, sendo elas:

- Falta de verificação na entrada de valores, para impedir a entrada de valores negativos.
- Falta de verificação para a entrada de ID, permitindo que o mesmo aceite dois IDs iguais.
- Não tem contenção para impedir a entrada de letras em lugar de número e número no lugar de letras.

Terceiro ponto negativo é que o código não foi tão comentado, um programa comentado facilita no entendimento do código principalmente se ele é feito em grupo ou se for necessário voltar a um programa antigo, os comentários facilitariam a retomada de onde parou.

➤ Modularização

A modularização é o ponto forte desse código, pois desde o início do código pode se ver que o mesmo é dividido em varias funções que atendem responsabilidade específicas (inserir, listar, consultar, alterar, excluir, comprar), tornando assim um código mais legível e organizado. Dessa forma facilita a manutenção dele, pois como cada função toma conta de uma parte quando for necessário arruma-la não prejudicará o resto do programa.

Ademais a utilização de funções auxiliares é um ponto forte do código, por exemplo a função “imprimirProduto” que armazena uma única vez o código de impressão e pode ser chamada por outras funções, dessa forma evitando um acúmulo desnecessário do mesmo código, deixando mais visual e facilitando na manutenção.

```
void consultarProduto(Produto *produtos, int cont) {
    int idProduto;
    printf("Informe o ID do produto que deseja consultar: ");
    scanf("%d", &idProduto);

    for (int i = 0; i < cont; i++) {
        if (produtos[i].id == idProduto) {
            imprimirProduto(&produtos[i]);
            return;
        }
    }
    printf("Produto inexistente.\n");
}
```

A utilização de ponteiro nas funções também é outro ponto positivo, pois ele permite que elas alterem os dados diretamente. Um caso dessa utilização é o contador de produtos “cont”, que é atualizado nas funções “inserirProduto”, “excluirProduto” e outras. Dessa forma mantendo sempre atualizado a contagem geral.

```
void inserirProduto(Produto *produtos, int *cont) {
    if (*cont < MAX_PRODUTOS) { // Verifica se há espaço no array
        Produto novoProduto;
```

Porém, ainda tem formas de melhorar essa modularização. Uma dessas melhorias seria na implementação de uma nova função focada na leitura de “strings”, pois há uma excessiva repetição do mesmo código (`scanf(" %[^\n]",)`). Outra melhoria seria as contenções de erros já mencionadas, a possibilidade de colocar uma letra em um local específico para número, verificação para valores negativos e IDs iguais. Tudo isso poderia ser implementado em funções específicas.

Contenção de erros

➤ Menu

No menu do programa existe contenção de erro para números, sendo valor fora das opções e mais de um número.

```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair
Escolha uma opcao: 8
Opcão invalida! Tente novamente.
```

Porém, não há contenção para letras e caracteres especiais. Outro ponto importante é que caso o usuário digite um número de espaço e digite outro, a primeira opção escolhida será executada com bug visual.

Usuário digitou letra ou caracteres especiais

```
7. Sair
Escolha uma opcao: Opcão 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: Opcão 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: Opcão invalida! Tente novamente.
```

Número espaço número

```
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 7
Informe o ID do produto: Informe o nome do produto: █
```

Entretanto caso o usuário escolha as opções 2 ou 6 de espaço e digite 7 o programa fecha automaticamente. As outras opções rodam o programa com ou sem bug visual.

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

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

➤ Opção 1

Não tem contenção de erro na entrada de letras onde deveria ser número e vice-versa, caso faça a listagem aparece o número no lugar onde deveria ser letra, mas se colocar letra no lugar de número o programa zera o valor.

Menu:

1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair

Escolha uma opção: 2

Lista de Produtos:

ID: 2, Nome: 9, Descricao: 10, Preco: 0.00, Estoque: 0

Além disso caso o usuário coloque letra no ID, preço do produto ou na quantidade ocorre bug visual.

Menu:

1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Comprar produto
7. Sair

Escolha uma opção: 1

Informe o ID do produto: oi

Informe o nome do produto: Informe a descrição do produto: caue

Informe o preço unitário: dez

Informe a quantidade disponível: 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 opção: Informe o ID do produto: Informe o nome do produto: Informe a descrição do produto:

```
E o a opção 1 não tem conteção para entrada de um mesmo ID.  
Escolha uma opcao: 1  
Informe o ID do produto: 2  
Informe o nome do produto: arroz  
Informe a descriçao do produto: bom  
Informe o preco unitario: 29.99  
Informe a quantidade disponivel: 10  
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: 1  
Informe o ID do produto: 2  
Informe o nome do produto: ■
```

➤ Opção 3,4,5,6

Contenção Simples de erro (Produto inexistente, Estoque insuficiente) e o mesmo problema com letras e caracteres especiais.

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

Código refatorado

➤ Correção da variável precoUnitario

A variável precoUnitario não seguia o requisito de ser “double”, por isso troquei o valor de float para double.

```
typedef struct {
    int id;
    char nome[NOME_MAX];
    char descricao[DESCRICAO_MAX];
    double precoUnitario; // Corrigido para double (antes era float)
    int qteDispo;
} Produto;
```

Dessa forma o código terá mais precisão para lidar com os preços dos produtos. Sendo muito útil para cálculos financeiros, que é a função desse código.

➤ Correção da entrada de IDs iguais

O código antigo permitia a entrada de IDs iguais, por isso corrigi esse erro adicionando uma nova função (produtoExiste) que percorre todo o array (produtos) e verifica se tem ID igual.

Depois chamei ela dentro da função (inserirProduto), dessa forma antes de inserir um novo produto tem que passar pela função para verificar se o ID não é o mesmo.

Função nova

```
int produtoExiste(Produto *produtos, int cont, int id) {
    for (int i = 0; i < cont; i++) {
        if (produtos[i].id == id) {
            return 1; // Produto já existe
        }
    }
    return 0; // Produto não encontrado
}
```

Chamando a função

```
void inserirProduto(Produto *produtos, int *cont) {
    if (*cont < MAX_PRODUTOS) { // Verifica se há espaço no array
        Produto novoProduto;
        int idProduto;

        printf("Informe o ID do produto: ");
        scanf("%d", &idProduto);

        if (produtoExiste(produtos, *cont, idProduto)) {
            printf("Erro: Produto com esse ID já existe.\n");
            return;
        }
    }
}
```

Contenção

```
Escolha uma opcao: 1
Informe o ID do produto: 2
Informe o nome do produto: arroz
Informe a descrição do produto: bom
Informe o preço unitário: 29.99
Informe a quantidade disponível: 10
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: 1
Informe o ID do produto: 2
Erro: Produto com esse ID já existe.
```

➤ Adição da opção “aplicar desconto”

A empresa pediu que o código tivesse a funcionalidade de aplicar desconto, porém o código que eu recebi não tem, por isso criei uma nova função (aplicarDesconto) que no final da aplicação atualiza o valor da variável “precoUnitário”.

```

void aplicarDesconto(Produto *produtos, int cont) {
    int idProduto;
    double desconto;

    printf("Informe o ID do produto para aplicar o desconto: ");
    scanf("%d", &idProduto);

    // Busca o produto pelo ID
    for (int i = 0; i < cont; i++) {
        if (produtos[i].id == idProduto) {
            printf("Informe o percentual de desconto (0-100): ");
            scanf("%lf", &desconto);

            if (desconto < 0 || desconto > 100) {
                printf("Desconto inválido. Deve ser entre 0 e 100.\n");
                return;
            }

            // Aplica o desconto
            double descontoValor = produtos[i].precoUnitario * (desconto / 100);
            produtos[i].precoUnitario -= descontoValor; // Atualiza o preço com o desconto

            printf("Desconto de %.2lf% aplicado com sucesso!\n", desconto);
            printf("Novo preço do produto: %.2f\n", produtos[i].precoUnitario);
            return;
        }
    }
    printf("Produto inexistente.\n");
}

```

No terminal

```

Escolha uma opção: 1
Informe o ID do produto: 2
Informe o nome do produto: arroz
Informe a descrição do produto: bom
Informe o preço unitário: 100
Informe a quantidade disponível: 10
Produto inserido com sucesso!

Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Aplicar desconto
7. Comprar produto
8. Sair
Escolha uma opção: 6
Informe o ID do produto para aplicar o desconto: 2
Informe o percentual de desconto (0-100): 10
Desconto de 10.00% aplicado com sucesso!
Novo preço do produto: 90.00

```

➤ Contenção de erro de letra/caracteres especiais

Como já mostrado, se o usuário digitar uma letra ou carácter especial o código fica em loop. Para resolver isso criei uma função auxiliar para a leitura(lerInteiroPositivo), pois o “scanf” não é capaz de separar o que é número de

letra, dessa forma acarretando não somente no loop, mas também na permissão de usar letra no lugar de número e vice-versa.

Resolução

```
int lerInteiroPositivo() {
    int num;
    while (scanf("%d", &num) != 1 || num <= 0) {
        while (getchar() != '\n'); // Limpa o buffer
        printf("Entrada inválida! Digite um número inteiro positivo: ");
    }
    return num;
}
```

No terminal

```
Menu:
1. Inserir produto
2. Listar produtos
3. Consultar produto
4. Alterar produto
5. Excluir produto
6. Aplicar desconto
7. Comprar produto
8. Sair
Escolha uma opção: k
Entrada inválida! Digite um número inteiro positivo: 
```

➤ Contenção de erro de número no lugar de letra e vice-versa

No código antigo era permitido a entrada de letras no lugar de número, essa permissão foi corrigida com a criação de funções auxiliares (lerString, stringValida, lerInteiroPositivo e lerDoublePositivo). Além disso, uma dessas funções auxiliares já corrige o erro de digitar valor negativo (lerInteiroPositivo), por exemplo ID negativo ou porcentagem negativa.

```
void lerString(char *str, int tamanho) {
    getchar(); // Limpa o buffer do '\n' que pode ficar do scanf anterior
    fgets(str, tamanho, stdin);
    str[strcspn(str, "\n")] = '\0'; // Remove o '\n' da string
}

// Função para verificar se a string contém apenas letras e espaços
int stringValida(char *str) {
    // Verifica se a string contém apenas letras e espaços
    for (int i = 0; str[i] != '\0'; i++) {
        if (!isalpha(str[i]) && !isspace(str[i])) {
            return 0; // Se encontrar um caractere não válido
        }
    }
    return 1; // Se a string for válida
}
```

```
// Funções auxiliares para Ler entradas com validação
int lerInteiroPositivo() {
    int num;
    while (scanf("%d", &num) != 1 || num <= 0) {
        while (getchar() != '\n'); // Limpa o buffer
        printf("Entrada inválida! Digite um número inteiro positivo: ");
    }
    return num;
}

double lerDoublePositivo() {
    double num;
    while (scanf("%lf", &num) != 1 || num <= 0) {
        while (getchar() != '\n'); // Limpa o buffer
        printf("Entrada inválida! Digite um número positivo válido: ");
    }
    return num;
}
```

No terminal

```
Escolha uma opção: 1
Informe o ID do produto: k
Entrada inválida! Digite um número inteiro positivo: -12
Entrada inválida! Digite um número inteiro positivo: 12
Informe o nome do produto: 9
Nome inválido! O nome deve conter apenas letras e espaços. Tente novamente: arroz
Informe a descrição do produto: 13
Descrição inválida! A descrição deve conter apenas letras e espaços. Tente novamente: gostoso e bom
Informe o preço unitário: -2
Entrada inválida! Digite um número positivo válido: 29.99
Informe a quantidade disponível: p
Entrada inválida! Digite um número inteiro positivo: 12
Produto inserido com sucesso!
```

➤ Correção dá entrada de um número espaço outro número

Para corrigir esse problema eu aproveitei a função (lerInteiroProduto), que já faz a contenção de número positivo, e adicionei uma condição de que ele aceite somente um número por vez e se ocorrer algo inesperado ele limpa o buffer, impedindo que aconteça o bug visual.

```
int lerInteiroPositivo() {
    int num;
    char ch;

    while (1) {
        if (scanf("%d", &num) != 1) {
            while ((ch = getchar()) != '\n' && ch != EOF); // Limpa o buffer até o final da linha
            printf("Entrada inválida! Digite um número inteiro positivo: ");
        } else {
            // Verifica se há mais caracteres após o número (como espaços ou outros números)
            if ((ch = getchar()) != '\n') {
                while (ch != '\n' && ch != EOF) {
                    ch = getchar(); // Limpa o restante da linha
                }
                printf("Entrada inválida! Digite apenas um número inteiro positivo: ");
            } else if (num <= 0) {
                printf("Entrada inválida! Digite um número inteiro positivo: ");
            } else {
                return num;
            }
        }
    }
}
```

Conclusão

Portanto como visto na análise acima, esse programa foi bem estruturado, seguia boa parte das requisições da empresa e tem uma boa modularização que divide o código em várias funções acarretando em um programa limpo e visual, além de impedir a utilização das mesmas linhas de código. Além de tudo, esse código segue “boas maneiras” como a utilização de camelCase e variáveis globais.

Contudo, têm melhorias e correções que deveriam ser feitas, por exemplo a utilização de “double” na variável (precoUnitario), a falta de contenção de erro, melhorias mínimas na modularização, a implementação de um código para executar desconto no produto e afins.

Todas essas correções foram adicionadas e podem ser vista no código refatorado , que eu subi no GitHub.