Jonathan Batista Bispo |Programação Modular e Estruturada Desafio N2-7| 22/11/2024

O código funciona perfeitamente no contexto do “mundo feliz” onde o usuário utiliza o código de maneira esperada pelo programador, contudo o código neste documento será analisado e pontuado todas as adesões, se foram bem executadas e as falhas o que poderiam ser evitadas ou até mesmo melhoradas, ao final de cada parágrafo será colocada em porcentagem o quanto tópicos requisitados foi aderido.

Modularização

Descrição:

Avaliação da estrutura do código em relação à separação em funções, organização em módulos suas reutilizações.

A modularização foi usada para a divisão das funções requisitadas pelas equipes, isso é a criação, alteração, consulta, exclusão e desconto relacionado aos produtos e/ou a listagem em ID e de produtos.

A modularização do código em questão está bem estruturada separando cada função aderindo o CRUD requisitado pela equipe, sendo mais prático ser usado pelo usuário e suas exigências. O uso de CamelCase foi bem colocado além das variáveis serem de fácil entendimento só de se observar seus nomes.

Pontos a Melhorar:

O único erro está na ordem que foi feito, visto que dentro do código é mencionado cada função que será usado visto a seguir

```
14  void cadastrarProduto(Produto *listaProdutos, int *contadorProduto);
15  void alterarProduto(Produto *listaProdutos, int ID, int contadorProduto);
16  void consultarProduto(Produto *listaProdutos, int ID, int contadorProduto);
17  void excluirProduto(Produto *listaProdutos, int ID, int *contadorProduto);
18  void imprimirDados(Produto *listaProdutos);
19  void venderProduto(Produto *listaProdutos, int ID, int quantidade, int contadorProduto);
20  void imprimirLista(Produto *listaProdutos, int contadorProduto);
21  void descontoProduto(Produto *listaProdutos, int ID, int desconto);
```

Esse método é útil quando os corpos das funções estão após o main( ), contudo todos os corpos estão antes do main( ), tornando essas menções irrelevantes para o código, será eliminado essas linhas de código apresentadas anteriormente na hora do refatoramento e descontado uma parte da adesão por esse erro.

Porcentagem de Adesão: **98%**

Implementação do CRUD (Create, Read, Update, Delete)

Descrição:

Avaliação da implementação das operações de criação, leitura, atualização e exclusão (CRUD), quando aplicável ao propósito do código. As funções de CRUD são funcionais, mas

somente no “mundo feliz”, visto que apresenta muitos erros e a falta de contenção de erros. Na questão de erros, ocorre uma gafe na parte de sua programação onde há uma troca da variável valorDoProduto por quantidadeEmEstoque visto a seguir:

```
printf("Digite o valor do produto: ");
scanf(" %lf", &produto -> valorDoProduto);
while(produto -> valorDoProduto < 0){
    printf("O valor digitado esta errado. So eh possivel adicionar valores positivos para os produtos.\n");
    printf("Digite novamente: ");
    scanf(" %d", &produto -> quantidadeEmEstoque);
}
```

Esse problema pode ser resolvido de forma muito simples, apenas trocando essas duas variáveis evitando assim um overflow por conta do valorDoProduto nunca ter a chance de ser atualizado.

```
printf("Digite o valor do produto: ");
while((scanf(" %lf", &produto -> valorDoProduto) != 1) || produto -> valorDoProduto < 0){
    printf("O valor digitado esta errado. So eh possivel adicionar valores positivos para os produtos.\n");
    printf("Digite novamente: ");
    scanf(" %lf", &produto -> valorDoProduto); // alteração da variável quantidadeEmEstoque para valorDoProduto
}
```

Enquanto a situação de falta de contenção de erro está no código requisitar apenas números positivos, onde caso o usuário digite uma letra, ocorre um overflow, o método para evitar isso é a adesão de um método que analisa os caracteres digitados, evitando erros de digitação aceitando somente números positivos conforme o ilustrado a seguir:

```
// Adesão de while evitando erros de digitação aceitando somente números positivos
while((scanf(" %lf", &produto->valorDoProduto) != 1) || produto-> quantidadeEmEstoque < 0){
    printf("A quantidade digitada esta errada. So eh possivel existir numeros positivos de produtos.\n");
    printf("Digite novamente: ");
    scanf(" %lf", &produto -> quantidadeEmEstoque);
}
```

O CRUD é funcional e traz exatamente o requerido pela empresa, com um ponto a mais por já implementar um sistema de automatização de IDs, evitando problemas de IDs iguais e não precisando se estender com contenções de erros.

Pontos a Melhorar:

Em sua maioria fica claro que o problema foi na hipótese do usuário digitar uma letra ao invés de número na aquisição de dígitos por meio do scanf, um problema que se torna grande quanto mais complexo é o código, decaindo muito a adesão ao tópico do CRUD.

Porcentagem de Adesão: **65%**

Uso do Struct

Descrição:

Verificação do uso consistente de Struct e se foram apropriados sua utilização. O uso de struct é uma ótima implementação, mesmo que já requerida para esse código, seu uso

precisa ser eficiente para uma melhor otimização para o uso de um banco de dados como deste contexto.

Pontos Positivos:

Os usos de tipos de variáveis que mais beneficiam suas variáveis como; int ID; char nomeProduto[50]; int quantidadeEmEstoque; double valorDoProduto. além de sua escrita tanto quanta das variáveis quanta variável do struct.

Pontos a Melhorar:

Deve se atentar o caso do nomeProduto[50] que possuí um número inserido diretamente, gerando um hardcode, o método para a solução disso é o uso ou a criação de uma diretiva e no caso do ID que quebra o camelCase sendo em maiúsculo não sendo uma variável global.

Porcentagem de Adesão: **98%**

Uso do Ponteiro

Descrição:

Foi verificada o uso dos ponteiros, que estão como uso para a redução de espaço armazenado na memória sobre um único valor dado de variável entre o main( ) e as demais funções. Sua implementação foi adequada extraindo o máximo que um ponteiro pode usufruir ao código.

Pontos Positivos:

Com uma boa execução do uso dos ponteiros, fica benéfico ao compilador na hora de se reservar a memória para a inserção de dados

Porcentagem de Adesão: 100%

Diretivas

Descrição:

Análise das diretivas de pré-processador utilizadas, como #define, #include. Foi colocado 3 bibliotecas no código, sendo o <stdio.h>; <string.h> e <math.h> e apenas uma variável global, o MAXPRODUTOS

Pontos Positivos:

Para melhor aproveitamento de strings e char, o uso da biblioteca do string.h foi essencial, além do uso de variáveis globais evitando hardcode.

Pontos a Melhorar:

Como não há cálculos que precisem da biblioteca <math.h> a sua inclusão ao código acaba se tornando inútil, sendo assim, seria mais prático não ter sua inclusão.

Porcentagem de Adesão: 99%

Nomes Significativos Para Variáveis

Descrição:

Para um código mais visual tanto para o próprio programador quanto para a equipe em si, o melhor uso na criação dos nomes é se atentar no que será inserido essa tal variável. O uso como citado anteriormente na parte da modularização é similar ao das variáveis, sendo muito adequado além do bom uso ao CamelCase.

Pontos Positivos:

As variáveis passam por seu nome a exata ideia do que serão propostas em meio ao código sendo mais prático certas manutenções e atualizações sem o uso excessivo de comentários explicando o que cada variável faz.

Porcentagem de Adesão: 100%

Conclusão Geral

Resumo da Análise:

Chegando na reta final fica claro que o código em questão funciona, de fato, perfeitamente no “mundo feliz”, contudo se saiu muito bem nos tópicos gerados por esse documento, é claro que estamos falando de um código feito para apresentar uma ideia hipotética, então muito dos defeitos apresentados levam uma certa experiência e tempo para serem analisados como uma aplicação de um contexto real.

Recomendações:

Os únicos pontos a se atentar são justamente as contenções de erros e a fixação de conceitos primordiais tanto para a programação como a linguagem C, visando sempre o bom uso de modularização, criação de variáveis, CRUD, ponteiros e struct para a expectativa de um bom programador.

Porcentagem Geral de Adesão do código: **94%**