FATEC IPIRANGA

PASTOR ENÉAS TOGNINI

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PROGRAMAÇÃO ESTRUTURADA E MODULAR

PROFESSOR CARLOS HENRIQUE VERISSIMO PEREIRA

MILENA MITIE AOKI

SÃO PAULO, SP
2024

SUMÁRIO

ATIVIDADE N2-7-ANÁLISE CRÍTICA DE CÓDIGO

INTRODUÇÃO

Este trabalho apresenta uma análise detalhada e a refatoração de um programa em linguagem C para a gestão de produtos em um sistema de estoque. O objetivo principal foi garantir a funcionalidade do código enquanto aprimorava sua robustez e usabilidade. Inicialmente, o programa apresentava limitações quanto ao tratamento de erros nas entradas do usuário, o que poderia levar a comportamentos inesperados e falhas na execução. A partir dessa observação, foram implementadas melhorias significativas, incluindo a validação de entradas numéricas e mensagens claras para o usuário em caso de erros. Além disso, foi realizada uma organização modular do código para facilitar a manutenção e a expansão futura do sistema.

A análise e as melhorias no código focaram na aplicação de boas práticas de programação, como o uso de funções para encapsular lógicas repetitivas, validação rigorosa de dados e comunicação eficaz com o usuário. Essas práticas são indispensáveis para desenvolver sistemas confiáveis e de fácil uso.

CÓDIGO ANALISADO

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#define MAXPRODUTOS 50

typedef struct{
    int ID;
    char nomeProduto[50];
    int quantidadeEmEstoque;
    double valorDoProduto;
} Produto;

void cadastrarProduto(Produto *listaProdutos, int *contadorProduto);
void alterarProduto(Produto *listaProdutos, int ID, int contadorProduto);
void consultarProduto(Produto *listaProdutos, int ID, int contadorProduto);
void excluirProduto(Produto *listaProdutos, int ID, int *contadorProduto);
void imprimirDados(Produto *listaProdutos);
void venderProduto(Produto *listaProdutos, int ID, int quantidade, int contadorProduto);
void imprimirLista(Produto *listaProdutos, int contadorProduto);
void descontoProduto(Produto *listaProdutos, int ID, int desconto);

void cadastrarProduto(Produto *listaProdutos, int *contadorProduto){
    Produto *produto = &listaProdutos[*contadorProduto];
    produto -> ID = *contadorProduto + 1;
    printf("\nDigite o nome do produto: ");
    scanf(" %[^\n]", produto -> nomeProduto);
    printf("Digite a quantidade em estoque do produto (Somente numeros naturais): ");
    scanf(" %i", &produto -> quantidadeEmEstoque);
    while(produto -> quantidadeEmEstoque < 0){
        printf("A quantidade digitada esta errada. So eh possivel existir numeros positivos de
produtos.\n");
        printf("Digite novamente: ");
        scanf(" %d", &produto -> quantidadeEmEstoque);
    }
    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);
```

```
    }

    (*contadorProduto)++;
    printf("Produto adicionado com sucesso!\n\n");
}

void imprimirDados(Produto *listaProdutos){
    printf("\nID: %d\n", listaProdutos -> ID);
    printf("Nome: %s\n", listaProdutos -> nomeProduto);
    printf("Quantidade em estoque: %d\n", listaProdutos -> quantidadeEmEstoque);
    printf("Valor do produto: %.2f\n\n", listaProdutos -> valorDoProduto);
}

void alterarProduto(Produto *listaProdutos, int ID, int contadorProduto){

    for(int i = 0; i < contadorProduto; i++){
        if(listaProdutos[i].ID == ID){
            printf("\nDados atuais do produto:\n");
            imprimirDados(&listaProdutos[i]);

            printf("\nDigite o nome do produto: ");
            scanf(" %[^\n]", listaProdutos[i].nomeProduto);
            printf("Digite a quantidade em estoque do produto: ");
            scanf(" %d", &listaProdutos[i].quantidadeEmEstoque);
            printf("Digite o valor do produto: ");
            scanf(" %lf", &listaProdutos[i].valorDoProduto);

            printf("Produto alterado com sucesso!\n\n");
            return;
        }
    }
    printf("Produto com o ID %d nao encontrado.\n\n", ID);
}

void consultarProduto(Produto *listaProdutos, int ID, int contadorProduto){
    for(int i = 0; i < contadorProduto; i++){
        if(listaProdutos[i].ID == ID){
        imprimirDados(&listaProdutos[i]);
        return;
        }
    }

    printf("Produto com o ID %d nao encontrado.\n\n", ID);
}

void excluirProduto(Produto *listaProdutos, int ID, int *contadorProduto){
    for(int i = 0; i < *contadorProduto; i++){
        if(listaProdutos[i].ID == ID){
```

```
            for(int j = i; j < *contadorProduto - 1; j++){
                strcpy(listaProdutos[j].nomeProduto, listaProdutos[j + 1].nomeProduto);
                listaProdutos[j].quantidadeEmEstoque = listaProdutos[j +
1].quantidadeEmEstoque;
                listaProdutos[j].valorDoProduto = listaProdutos[j + 1].valorDoProduto;
            }
            (*contadorProduto)--;
            printf("Produto excluido com sucesso!\n\n");

            return;
        }
    }
    printf("Produto com o ID %d nao encontrado.\n\n", ID);
}

void venderProduto(Produto *listaProdutos, int ID, int quantidade, int contadorProduto){
    for(int i = 0; i < contadorProduto; i++){
        if(listaProdutos[i].ID == ID){
            if (listaProdutos[i].quantidadeEmEstoque >= quantidade) {
                printf("Preco da venda: %.2f\n", listaProdutos[i].valorDoProduto * quantidade);
                listaProdutos[i].quantidadeEmEstoque -= quantidade;
                printf("Produto vendido com sucesso!\n\n");
                return;
            }
            else {
                printf("Estoque insuficiente para realizar a venda.\n\n");
            }
        }
    }
    printf("Produto com o ID %d nao encontrado.\n\n", ID);
}

void imprimirLista(Produto *listaProdutos, int contadorProduto){
    printf("\n---------- Lista de todos os produtos cadastrados ----------\n\n");
    for(int i = 0; i < contadorProduto; i++){
        printf("ID: %d      Nome: %s\n", listaProdutos[i].ID, listaProdutos[i].nomeProduto);
    }
    printf("\n");
}

void descontoProduto(Produto *listaProdutos, int ID, int contadorProduto){
    for(int i = 0; i < contadorProduto; i++){
        if(listaProdutos[i].ID == ID){

            double desconto;

            printf("\nDigite o desconto que será dado ao produto\n");
            printf("Exemplo: Se for diminuir o preco em 45/100 digite somente 45.\n");
```

```
            scanf("%lf", &desconto);

            listaProdutos[i].valorDoProduto = listaProdutos[i].valorDoProduto -
(listaProdutos[i].valorDoProduto * (desconto / 100));
            printf("Novo valor do produto: %.2f\n", listaProdutos[i].valorDoProduto);
            printf("Desconto aplicado com sucesso!\n\n");
            return;
        }
    }

    printf("Produto com o ID %d não encontrado.\n\n", ID);
}

int main(){

    Produto listaProdutos[MAXPRODUTOS];
    int contadorProduto = 0;
    int opcao, ID, quantidade;

    do{
        printf("------ * MENU * ------\n");
        printf("1. Cadastrar produto\n");
        printf("2. Alterar dados do produto\n");
        printf("3. Consultar produto\n");
        printf("4. Excluir Produto\n");
        printf("5. Consultar lista de produtos\n");
        printf("6. Vender produto\n");
        printf("7. Dar desconto a um produto\n");
        printf("8. Encerrar o programa\n");
        printf("Escolha uma opcao: ");
        scanf(" %d", &opcao);

        switch(opcao){
            case 1: cadastrarProduto(listaProdutos, &contadorProduto);
                    break;

            case 2: printf("\nDigite o ID do produto que será alterado: ");
                    scanf(" %d", &ID);
                    alterarProduto(listaProdutos, ID, contadorProduto);
                    break;

            case 3: printf("\nDigite o ID do produto que deseja consultar: ");
                    scanf(" %d", &ID);
                    consultarProduto(listaProdutos, ID, contadorProduto);
                    break;

            case 4: printf("\nDigite o ID do produto que deseja excluir: ");
                    scanf(" %d", &ID);
```

```
            excluirProduto(listaProdutos, ID, &contadorProduto);
            break;

        case 5: imprimirLista(listaProdutos, contadorProduto);
            break;

        case 6: printf("\nDigite o ID do produto que deseja vender: ");
            scanf(" %d", &ID);
            printf("Digite a quantidade que será vendida: ");
            scanf(" %d", &quantidade);
            while(quantidade <= 0){
               printf("A quantidade a ser vendida deve ser maior ou igual a 1. Entre
novamente com o valor: ");
               scanf("%d", &quantidade);
            }
            venderProduto(listaProdutos, ID, quantidade, contadorProduto);
            break;

        case 7: printf("\nDigite o ID do produto que recebera desconto: ");
            scanf(" %d", &ID);
            descontoProduto(listaProdutos, ID, contadorProduto);
            break;

        case 8: printf("\nObrigado por usar o nosso programa na sua loja!\n");
            printf("Encerrando a sua execucao...");
            break;

        default: printf("\nO valor digitado nao eh valido. Digite novamente: \n\n");
             break;
      }

   }while(opcao != 8);

   return 0;
}
```

1. MODULARIZAÇÃO

O código apresenta uma boa divisão de responsabilidades, com funções separadas para cada operação do CRUD (incluir, alterar, consultar e excluir produtos) e funcionalidades adicionais como aplicar descontos e realizar vendas. Os dados são manipulados exclusivamente por meio de ponteiros, conforme os requisitos.

No entanto, existem redundâncias no código (ex.: busca de produtos por ID repetida em várias funções) que poderiam ser eliminadas com a criação de funções auxiliares.

2. ELEMENTOS CONCEITUAIS

Estrutura de dados (Struct): O programa define uma `struct Produto`, que armazena as propriedades do produto de forma clara e organizada:

- `ID`: Identificador único para cada produto.
- `nomeProduto`: Nome do produto (string).
- `quantidadeEmEstoque`: Quantidade disponível no estoque (int).
- `valorDoProduto`: Preço do produto (double).

Ponteiros: Todas as funções manipulam a lista de produtos e suas propriedades utilizando ponteiros, o que permite alterações diretas na estrutura.

CRUD: O código implementa corretamente as operações de Create (cadastrar), Read (consultar), Update (alterar) e Delete (excluir), além de outras funcionalidades específicas do domínio da loja.

3. ELEMENTOS DE NEGÓCIO (REQUISITOS)

O código atende aos seguintes requisitos descritos na especificação:

- **Cadastro de produtos:** Implementado na função `cadastrarProduto`.

- **Consulta por código:** Implementado na função `consultarProduto`.
- **Alteração por ID:** Implementado na função `alterarProduto`.
- **Exclusão por ID:** Implementado na função `excluirProduto`.
- **Vender produto:** Implementado na função `venderProduto`.
- **Aplicar desconto:** Implementado na função `descontoProduto`.

Adicionalmente, o programa inclui funcionalidades extras, como validação de entradas (quantidade e preço), impressão de lista de produtos e controle de estoque.

4. REGRAS EXPLÍCITAS

Regras diretamente codificadas no programa:

1. **ID único:** Cada produto recebe um ID incremental ao ser cadastrado.
2. **Validação de entradas:**
    - Quantidade em estoque não pode ser negativa.
    - Preço do produto não pode ser negativo.
3. **Venda de produto:**
    - A venda só ocorre se houver estoque suficiente.
4. **Desconto:**
    - O desconto é aplicado como uma porcentagem do preço original.
5. **Limite de produtos:** O número máximo de produtos é limitado por `MAXPRODUTOS`.

5. REGRAS IMPLÍCITAS

Regras não explicitamente descritas, mas inferidas pelo comportamento do código:

1. **ID não reaproveitado:** Após excluir um produto, seus IDs subsequentes permanecem inalterados.
2. **Persistência em memória:** O programa não utiliza arquivos ou banco de dados, então os dados são perdidos ao encerrar o programa.
3. **Ordem fixa dos produtos:** A exclusão de um produto desloca os produtos subsequentes na lista, mas mantém a ordem de cadastro.

CÓDIGO REFATORADO

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#define MAXPRODUTOS 50

// Estrutura de Produto
typedef struct {
    int ID;
    char nomeProduto[50];
    int quantidadeEmEstoque;
    double valorDoProduto;
} Produto;

// Prototipação das funções
void limparBuffer();
int entradaIntSegura(const char *mensagem);
double entradaDoubleSegura(const char *mensagem);
void entradaStringSegura(char *destino, int tamanho, const char *mensagem);
void cadastrarProduto(Produto *listaProdutos, int *contadorProduto);
void imprimirDados(const Produto *produto);
void alterarProduto(Produto *listaProdutos, int ID, int contadorProduto);
void consultarProduto(const Produto *listaProdutos, int ID, int contadorProduto);
void excluirProduto(Produto *listaProdutos, int ID, int *contadorProduto);
void venderProduto(Produto *listaProdutos, int ID, int quantidade, int
contadorProduto);
```

```
void imprimirLista(const Produto *listaProdutos, int contadorProduto);
void descontoProduto(Produto *listaProdutos, int ID, int contadorProduto);

// Função para limpar o buffer do teclado
void limparBuffer() {
    int c;
    while ((c = getchar()) != '\n' && c != EOF);
}

// Função para ler um número inteiro com validação
int entradaIntSegura(const char *mensagem) {
    int valor;
    char buffer[20];
    while (1) {
        printf("%s", mensagem);
        if (fgets(buffer, sizeof(buffer), stdin) != NULL && sscanf(buffer, "%d", &valor) ==
1) {
            return valor;
        }
        printf("Entrada inválida! Digite um número inteiro.\n");
        limparBuffer();
    }
}

// Função para ler um número double com validação
double entradaDoubleSegura(const char *mensagem) {
    double valor;
    char buffer[50];
```

```
    while (1) {
        printf("%s", mensagem);
        if (fgets(buffer, sizeof(buffer), stdin) != NULL && sscanf(buffer, "%lf", &valor) ==
1) {
            return valor;
        }
        printf("Entrada inválida! Digite um valor numérico válido.\n");
        limparBuffer();
    }
}

// Função para ler uma string com validação
void entradaStringSegura(char *destino, int tamanho, const char *mensagem) {
    printf("%s", mensagem);
    if (fgets(destino, tamanho, stdin)) {
        size_t len = strlen(destino);
        if (len > 0 && destino[len - 1] == '\n') {
            destino[len - 1] = '\0';
        }
    }
}

// Função para cadastrar um produto
void cadastrarProduto(Produto *listaProdutos, int *contadorProduto) {
    if (*contadorProduto >= MAXPRODUTOS) {
        printf("Número máximo de produtos cadastrados atingido.\n");
        return;
    }
```

```
    Produto *produto = &listaProdutos[*contadorProduto];
    produto->ID = *contadorProduto + 1;

    entradaStringSegura(produto->nomeProduto, sizeof(produto->nomeProduto),
"\nDigite o nome do produto: ");

    produto->quantidadeEmEstoque = entradaIntSegura("Digite a quantidade em
estoque (número natural): ");
    while (produto->quantidadeEmEstoque < 0) {
        printf("A quantidade não pode ser negativa. Tente novamente.\n");
        produto->quantidadeEmEstoque = entradaIntSegura("Digite a quantidade em
estoque (número natural): ");
    }

    produto->valorDoProduto = entradaDoubleSegura("Digite o valor do produto: ");
    while (produto->valorDoProduto < 0) {
        printf("O valor não pode ser negativo. Tente novamente.\n");
        produto->valorDoProduto = entradaDoubleSegura("Digite o valor do produto: ");
    }

    (*contadorProduto)++;
    printf("Produto cadastrado com sucesso!\n");
}

// Função para imprimir os dados de um produto
void imprimirDados(const Produto *produto) {
    printf("\nID: %d\n", produto->ID);
    printf("Nome: %s\n", produto->nomeProduto);
```

```
    printf("Quantidade em estoque: %d\n", produto->quantidadeEmEstoque);
    printf("Valor do produto: %.2f\n\n", produto->valorDoProduto);
}

// Função para alterar um produto
void alterarProduto(Produto *listaProdutos, int ID, int contadorProduto) {
    for (int i = 0; i < contadorProduto; i++) {
        if (listaProdutos[i].ID == ID) {
            printf("\nDados atuais do produto:\n");
            imprimirDados(&listaProdutos[i]);

            entradaStringSegura(listaProdutos[i].nomeProduto,
sizeof(listaProdutos[i].nomeProduto), "Digite o novo nome do produto: ");
            listaProdutos[i].quantidadeEmEstoque = entradaIntSegura("Digite a nova
quantidade em estoque: ");
            listaProdutos[i].valorDoProduto = entradaDoubleSegura("Digite o novo valor
do produto: ");

            printf("Produto alterado com sucesso!\n");
            return;
        }
    }
    printf("Produto com ID %d não encontrado.\n", ID);
}

// Função para consultar um produto
void consultarProduto(const Produto *listaProdutos, int ID, int contadorProduto) {
    for (int i = 0; i < contadorProduto; i++) {
        if (listaProdutos[i].ID == ID) {
```

```
            imprimirDados(&listaProdutos[i]);
            return;
        }
    }
    printf("Produto com ID %d não encontrado.\n", ID);
}

// Função para excluir um produto
void excluirProduto(Produto *listaProdutos, int ID, int *contadorProduto) {
    for (int i = 0; i < *contadorProduto; i++) {
        if (listaProdutos[i].ID == ID) {
            for (int j = i; j < *contadorProduto - 1; j++) {
                listaProdutos[j] = listaProdutos[j + 1];
            }
            (*contadorProduto)--;
            printf("Produto excluído com sucesso!\n");
            return;
        }
    }
    printf("Produto com ID %d não encontrado.\n", ID);
}

// Função para vender um produto
void venderProduto(Produto *listaProdutos, int ID, int quantidade, int
contadorProduto) {
    for (int i = 0; i < contadorProduto; i++) {
        if (listaProdutos[i].ID == ID) {
            if (listaProdutos[i].quantidadeEmEstoque >= quantidade) {
```

```
            printf("Preço da venda: %.2f\n", listaProdutos[i].valorDoProduto *
quantidade);
            listaProdutos[i].quantidadeEmEstoque -= quantidade;
            printf("Produto vendido com sucesso!\n");
            return;
        } else {
            printf("Estoque insuficiente.\n");
            return;
        }
    }
    printf("Produto com ID %d não encontrado.\n", ID);
}

// Função para listar todos os produtos
void imprimirLista(const Produto *listaProdutos, int contadorProduto) {
    printf("\n--- Lista de Produtos ---\n");
    for (int i = 0; i < contadorProduto; i++) {
        printf("ID: %d | Nome: %s\n", listaProdutos[i].ID, listaProdutos[i].nomeProduto);
    }
    printf("\n");
}

// Função para aplicar desconto a um produto
void descontoProduto(Produto *listaProdutos, int ID, int contadorProduto) {
    for (int i = 0; i < contadorProduto; i++) {
        if (listaProdutos[i].ID == ID) {
            double desconto = entradaDoubleSegura("Digite o desconto em
porcentagem (0-100): ");
```

```
            while (desconto < 0 || desconto > 100) {
                printf("Porcentagem inválida. Tente novamente.\n");
                desconto = entradaDoubleSegura("Digite o desconto em porcentagem
(0-100): ");
            }
            listaProdutos[i].valorDoProduto -= listaProdutos[i].valorDoProduto *
(desconto / 100);
            printf("Novo valor do produto: %.2f\n", listaProdutos[i].valorDoProduto);
            return;
        }
    }
    printf("Produto com ID %d não encontrado.\n", ID);
}

// Função principal
int main() {
    Produto listaProdutos[MAXPRODUTOS];
    int contadorProduto = 0;

    while (1) {
        printf("\n--- MENU ---\n");
        printf("1. Cadastrar produto\n");
        printf("2. Alterar produto\n");
        printf("3. Consultar produto\n");
        printf("4. Excluir produto\n");
        printf("5. Listar produtos\n");
        printf("6. Vender produto\n");
        printf("7. Aplicar desconto\n");
        printf("8. Sair\n");
```

```
int opcao = entradaIntSegura("Escolha uma opção: ");

switch (opcao) {
    case 1: cadastrarProduto(listaProdutos, &contadorProduto); break;
    case 2: {
        int ID = entradaIntSegura("Digite o ID do produto: ");
        alterarProduto(listaProdutos, ID, contadorProduto);
        break;
    }
    case 3: {
        int ID = entradaIntSegura("Digite o ID do produto: ");
        consultarProduto(listaProdutos, ID, contadorProduto);
        break;
    }
    case 4: {
        int ID = entradaIntSegura("Digite o ID do produto: ");
        excluirProduto(listaProdutos, ID, &contadorProduto);
        break;
    }
    case 5: imprimirLista(listaProdutos, contadorProduto); break;
    case 6: {
        int ID = entradaIntSegura("Digite o ID do produto: ");
        int quantidade = entradaIntSegura("Digite a quantidade a ser vendida: ");
        venderProduto(listaProdutos, ID, quantidade, contadorProduto);
        break;
    }
    case 7: {
        int ID = entradaIntSegura("Digite o ID do produto: ");
```

```
                descontoProduto(listaProdutos, ID, contadorProduto);
                break;
            }
            case 8: printf("Encerrando programa.\n"); return 0;
            default: printf("Opção inválida! Tente novamente.\n");
        }
    }
}
```

CONCLUSÃO

O processo de refatoração e análise do código resultou em um programa significativamente mais eficiente e seguro. A inclusão de validações para entradas de dados reduziu consideravelmente o risco de erros causados por entradas inválidas, melhorando a experiência do usuário. A modularização aprimorada facilitou a organização e a leitura do código, tornando-o mais sustentável e adaptável a futuras alterações ou expansões.

Este trabalho demonstra a importância de investir em boas práticas de desenvolvimento, como a validação de dados e a modularização, em sistemas computacionais. Além disso, reforça a relevância de uma abordagem centrada no usuário, evidenciada pelas mensagens claras de erro e orientações no menu. Com isso, o programa está pronto para atender aos requisitos do sistema de gerenciamento de estoque, oferecendo uma solução funcional, confiável e de alta qualidade.