Análise crítica do código 7

Por: Diego Lopes Sakata **data: 25/11/2024**

Fatec Ipiranga – 2º Semestre – Analise e Desenvolvimento de Sistemas

Sumário

1. Código em questão:

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

// Estrutura do produto
typedef struct {
```

```
    int ID;
    char nomeProd[50];
    int qntdEstoque;
    double valorProduto;
} Produto;

// Função para criar um produto
Produto* SetProduto(int ID, char* nomeProd, int qntdEstoque, double valorProduto) {
    Produto* prod = (Produto*)malloc(sizeof(Produto));
    if (prod != NULL) {
        prod->ID = ID;
        strcpy(prod->nomeProd, nomeProd);
        prod->qntdEstoque = qntdEstoque;
        prod->valorProduto = valorProduto;
    }
    return prod;
}

// Função para imprimir os detalhes de um produto
void imprimeProduto(Produto* prod) {
    if (prod != NULL) {
        printf("\n*--------------------------------*");
        printf("\n * Id: %d", prod->ID);
        printf("\n * Nome Produto: %s", prod->nomeProd);
        printf("\n * Quantidade Disponível: %d", prod->qntdEstoque);
        printf("\n * Preço/Valor: %.2f", prod->valorProduto);
        printf("\n*--------------------------------*");
    }
}

// Função para incluir um novo produto
void incluirProduto(Produto* produtos[], int* totalProdutos) {
```

```
    int id, quantidade, valor id = 0;
    double preco;
    char nome[50];

    printf("Digite o ID do produto: ");
    scanf("%d", &id);
    printf("Digite o nome do produto: ");
    scanf(" %[^\n]s", nome); // lê até o enter
    printf("Digite a quantidade em estoque: ");
    scanf("%d", &quantidade);
    printf("Digite o valor do produto: ");
    scanf("%lf", &preco);

    produtos[*totalProdutos] = SetProduto(id, nome, quantidade, preco);
    (*totalProdutos)++;
    printf("Produto adicionado com sucesso!\n");
}

// Função para alterar um produto existente por ID
void alterarProduto(Produto* produtos[], int totalProdutos) {
    int id, novaQuantidade;
    double novoPreco;
    char novoNome[50];
    bool encontrado = false;

    printf("Digite o ID do produto que deseja alterar: ");
    scanf("%d", &id);

    for (int i = 0; i < totalProdutos; i++) {
        if (produtos[i]->ID == id) {
            printf("Digite o novo nome do produto: ");
            scanf(" %[^\n]s", novoNome);
```

```
            printf("Digite a nova quantidade em estoque: ");
            scanf("%d", &novaQuantidade);
            printf("Digite o novo valor do produto: ");
            scanf("%lf", &novoPreco);

            strcpy(produtos[i]->nomeProd, novoNome);
            produtos[i]->qntdEstoque = novaQuantidade;
            produtos[i]->valorProduto = novoPreco;
            printf("Produto alterado com sucesso!\n");
            encontrado = true;
            break;
        }
    }
    if (!encontrado) {
        printf("Produto com ID %d não encontrado.\n", id);
    }
}

// Função para consultar um produto por ID
void consultarProduto(Produto* produtos[], int totalProdutos) {
    int id;
    bool encontrado = false;

    printf("Digite o ID do produto que deseja consultar: ");
    scanf("%d", &id);

    for (int i = 0; i < totalProdutos; i++) {
        if (produtos[i]->ID == id) {
            imprimeProduto(produtos[i]);
            encontrado = true;
            break;
        }
```

```
    }
    if (!encontrado) {
        printf("Produto com ID %d não encontrado.\n", id);
    }
}

// Função para excluir um produto por ID
void excluirProduto(Produto* produtos[], int* totalProdutos) {
    int id;
    bool encontrado = false;

    printf("Digite o ID do produto que deseja excluir: ");
    scanf("%d", &id);

    for (int i = 0; i < *totalProdutos; i++) {
        if (produtos[i]->ID == id) {
            free(produtos[i]); // libera memória do produto
            for (int j = i; j < *totalProdutos - 1; j++) {
                produtos[j] = produtos[j + 1];
            }
            (*totalProdutos)--;
            printf("Produto excluído com sucesso!\n");
            encontrado = true;
            break;
        }
    }
    if (!encontrado) {
        printf("Produto com ID %d não encontrado.\n", id);
    }
}

// Função para imprimir todos os produtos
```

```
void imprimirDadosProd(Produto* produtos[], int totalProdutos) {
    if (totalProdutos == 0) {
        printf("Nenhum produto cadastrado.\n");
    } else {
        for (int i = 0; i < totalProdutos; i++) {
            imprimeProduto(produtos[i]);
        }
    }
}

// Função para aplicar desconto ao produto
void aplicarDesconto(Produto* produtos[], int totalProdutos) {
    int id;
    double desconto;
    bool encontrado = false;

    printf("Digite o ID do produto para aplicar o desconto: ");
    scanf("%d", &id);
    printf("Digite o percentual de desconto (exemplo, para 10%%, digite 10): ");
    scanf("%lf", &desconto);

    for (int i = 0; i < totalProdutos; i++) {
        if (produtos[i]->ID == id) {
            produtos[i]->valorProduto *= (1 - desconto / 100.0);
            printf("Desconto aplicado com sucesso! Novo valor: %.2f\n", produtos[i]-
>valorProduto);
            encontrado = true;
            break;
        }
    }
    if (!encontrado) {
        printf("Produto com ID %d não encontrado.\n", id);
    }
```

```
}

int main() {
    Produto* produtos[10];
    int totalProdutos = 0;
    int opcao;

    printf("Seja bem-vindo à loja de produtos eletrônicos!\n");

    while (true) {
        printf("\nMenu de opções:\n1 - Incluir produto\n2 - Alterar produto por ID\n3 - Consultar produto por ID\n4 - Excluir produto\n");
        printf("5 - Imprimir todos os produtos\n6 - Aplicar desconto em produto\n7 - Sair\nEscolha uma opção: ");
        scanf("%d", &opcao);

        switch (opcao) {
            case 1:
                incluirProduto(produtos, &totalProdutos);
                break;
            case 2:
                alterarProduto(produtos, totalProdutos);
                break;
            case 3:
                consultarProduto(produtos, totalProdutos);
                break;
            case 4:
                excluirProduto(produtos, &totalProdutos);
                break;
            case 5:
                imprimirDadosProd(produtos, totalProdutos);
                break;
            case 6:
```

```
            aplicarDesconto(produtos, totalProdutos);
            break;
        case 7:
            printf("Obrigado por usar o sistema da loja!\n");
            for (int i = 0; i < totalProdutos; i++) {
                free(produtos[i]); // liberar memória dos produtos ao sair
            }
            return 0;
        default:
            printf("Opção inválida!\n");
            break;
    }
  }
}
```

2. Análise do funcionamento do código e de seus objetivos

2.1. Como o código funciona?

2.1.1. Estrutura de Dados

- **struct Produto**: Define um produto com os seguintes campos:
  - ID: Identificador único do produto.
  - nomeProd: Nome do produto.
  - qntdEstoque: Quantidade disponível em estoque.
  - valorProduto: Preço do produto.

2.1.2. Funções

As funções manipulam produtos armazenados em um array de ponteiros.

Função SetProduto

Cria dinamicamente um novo produto:

- Recebe os atributos do produto (ID, nome, quantidade, valor).
- Aloca memória com malloc e inicializa os valores.
- Retorna o ponteiro para o novo produto.

Função imprimeProduto

Exibe as informações de um produto formatadas.

Função incluirProduto

Adiciona um novo produto ao array:

- Solicita informações ao usuário.
- Cria o produto com SetProduto e armazena no array.
- Incrementa totalProdutos.

Função alterarProduto

Permite editar um produto existente:

- Busca o produto pelo ID.
- Solicita os novos valores e os atualiza.
- Exibe mensagem se o produto foi encontrado ou não.

Função consultarProduto

Exibe os detalhes de um produto com base no ID.

Função excluirProduto

Remove um produto:

- Busca pelo ID e libera sua memória com free.
- Realiza o "shift" do array para preencher a lacuna deixada.
- Decrementa totalProdutos.

Função imprimirDadosProd

Imprime todos os produtos cadastrados:

- Percorre o array e chama imprimeProduto para cada produto.

Função aplicarDesconto

Aplica um desconto ao valor de um produto específico:

- Busca pelo ID.
- Atualiza o valor do produto com base no percentual de desconto informado.

2.1.3. Função Principal (main)

A função principal oferece um menu de opções interativo:

- **Controle de Produtos**: totalProdutos controla a quantidade de produtos cadastrados.
- O menu é implementado com um while (true) e um switch para tratar cada opção:
    1. **Incluir Produto**
    2. **Alterar Produto por ID**

3. **Consultar Produto por ID**
4. **Excluir Produto**
5. **Imprimir Todos os Produtos**
6. **Aplicar Desconto**
7. **Sair**
    - Libera a memória de todos os produtos e finaliza o programa.

2.2. Objetivo do programa

O seu principal objetivo é auxiliar os trabalhadores de uma loja de produtos eletrônicos. Sendo capaz de armazenar os dados dos produtos e aplicando descontos de maneira simples e rápida nos preços dos mesmos.

2.3. O objetivo foi atingido?

Sim, o programa consegue realizar essas duas atividades principais. Entretanto, não é de maneira perfeita, existem alguns erros que podem acontecer durante a execução do usuário que não foram previstos. Um exemplo de erro seria a possibilidade de aplicar mais de 100% de desconto em um produto, o deixando com o valor negativo. Irei explicar melhor esses erros no tópico de “Melhorias”, que estará adiante.

2.4. O que foi pedido pelo cliente para a criação do programa?

Criar um programa em C que armazene dados de um produto e permita que o usuário aplique descontos aos seus preços. É necessário criar funções CRUD, uma função para imprimir todos os produtos cadastrados e a função que realizará a aplicação dos descontos. Utilizar ponteiros na construção de todas as funções e um struct para o array de produtos.

2.5. O programa atendeu todos os requisitos?

De maneira simplificada, sim, o programa conseguiu atingir os requisitos pedidos pelo seu cliente e executa as funções com sucesso. Mas existem alguns problemas, que podem “quebrar” o código durante a sua execução, que precisam ser melhorados. Um exemplo seria permitir a entrada de letras em um campo que deveria aceitar somente valores inteiros.

2.6. Qual é a qualidade do código?

O código é organizado, com funções separadas para cada operação, e tem nomes de variáveis autoexplicativos, utilizando o Camel Case. Além disso ainda apresenta comentários, que ajudam a entender as funções. A lógica também é clara e fácil de entender, mas poderia ser mais modularizado para evitar repetições. Ou seja, o código apresenta uma boa qualidade, mas ainda é necessário realizar alterações para deixar o código melhor.

2.7. Feedback geral

O código apresenta muitos pontos fortes, como por exemplo: Ser bem organizado e funcional; utiliza memória dinâmica para gerenciar produtos; oferece um menu interativo e várias opções úteis. Mas ainda existem áreas que necessitam de melhorias e algumas funções podem ser adicionadas para facilitar a sua utilização pelo usuário.

3. Pontos de melhorias e alterações

3.1. Modularização

O programa está bem modularizado, cada função realiza uma atividade especifica. Porém, ainda é possível melhorar, criando mais delas para evitar repetições que ocorrem no código e algumas que trabalhem somente com I/O. Também é possível criar novas funções com o objetivo de realizar novas ações e melhorar o programa.

3.2. Pontos de melhorias

3.2.1. Elementos conceituais

Os elementos abordados nessa parte são aqueles que não irão adicionar algo novo no programa para o usuário, mas irá melhorar a sua execução prevendo erros e evitando que eles aconteçam. Os problemas que estão acontecendo no código relacionados a esses elementos são:

- O programa está quebrando quando o usuário entra com uma letra no menu, no lugar de um número.
- Está aceitando a entrada de letras nos locais que deveriam ser números, tanto inteiros quanto double. Assim quebrando o código.
- Permite a inclusão de produtos com o mesmo ID.
- Não existem funções que trabalhem somente com I/O.

As melhorias que precisam ser realizadas são:

- Criar uma função para verificar se os valores das entradas de dados são permitidos, por exemplo, uma função que verifica se o caractere digitado é um número, caso for uma letra ou um símbolo, ela pede outra entrada de dados.
- Criar uma função que verifica se já existem produtos com determinado ID, ou fazer com o que o ID não seja entrado pelo usuário, mas sim incrementado pelo próprio programa. E ao realizar a exclusão, o programa iria ajeitar todos os IDs para não ficar um espaço vazio entre eles.
- Criar novas funções que trabalhem somente com I/O.

3.2.2. Elementos de negócio

Os elementos de negócio são aqueles requisitados pelo contratante e aqueles que devem ser subentendidos, para que uma entrada errada não comprometa a atividade da loja. Os problemas relacionados a esses elementos são:

- É possível adicionar um desconto maior do que 100% do valor do produto, deixando, assim, o mesmo com um valor negativo.
- É possível adicionar um desconto negativo no produto, aumentando o seu valor.
- O programa está permitindo a entrada de valores negativos em áreas que não deveria, como na parte de ”valor” e “quantidade” durante o cadastro de um produto.

As melhorias que precisam ser realizadas são:

- Delimitar a quantidade de desconto possível para um produto (Entre 0% e 100%).
- Delimitar que as entradas de valores que deveriam ser positivos, aceitem somente valores positivos.

Algumas melhorias que podem ser acrescentadas para melhorar o trabalho do usuário são:

- Adicionar uma função para alterar somente a quantidade em estoque de um determinado produto, para quando chegar um novo estoque, ele não precise alterar todos os dados do produto, apenas a quantidade.
- Adicionar a função de vender produtos, mostrando o valor da venda e diminuindo a quantidade no estoque.
- Acrescentar uma função só para aumentar o preço dos produtos.

3.3. Alterações realizadas no código

3.3.1. Novas funções criadas

Função “lerString”: Tem como único objetivo receber inputs de strings.

```
// Função para ler uma string
void lerString(char* string, int tamanho) {
    scanf(" %[^\n]s", string);
}
```

Função “lerInt”: Tem como único objetivo receber inputs de valores inteiros e realizar a validação desse valor, ou seja, caso digitem uma letra ou um símbolo, ele pedirá por uma nova entrada.

```
53  // Função para ler um inteiro
54  int lerInt() {
55      int valor;
56      bool valid = false;
57      while (!valid) {
58          if (scanf("%d", &valor) != 1) {
59              printf("Entrada inválida. Por favor, insira um número.\n");
60              while (getchar() != '\n'); // Limpa o buffer de entrada
61          } else {
62              valid = true;
63          }
64      }
65      return valor;
66  }
67
```

Função “lerDouble: Seu objetivo é praticamente o mesmo da função anterior, mas no lugar de valores inteiros, são double.

```
68  // Função para ler um double
69  double lerDouble() {
70      double valor;
71      bool valid = false;
72      while (!valid) {
73          if (scanf("%lf", &valor) != 1) {
74              printf("Entrada inválida. Por favor, insira um número.\n");
75              while (getchar() != '\n'); // Limpa o buffer de entrada
76          } else {
77              valid = true;
78          }
79      }
80      return valor;
81  }
82
```

Função “lerIntPositivo”: Nos locais que são obrigatórios a entrada de valores inteiros positivos, essa função garante que se o usuário entrar com um valor negativo, o programa pede mais uma entrada.

```
84  // Função para ler um inteiro positivo
85  int lerIntPositivo() {
86      int valor;
87      do {
88          valor = lerInt();
89          if (valor < 0) {
90              printf("Por favor, insira um valor positivo: ");
91          }
92      } while (valor < 0);
93      return valor;
94  }
95
```

Função “lerDoublePositivo”: Mesma função da anterior, porém com valores double.

```c
// Função para ler um double positivo
double lerDoublePositivo() {
    double valor;
    do {
        valor = lerDouble();
        if (valor < 0) {
            printf("Por favor, insira um valor positivo: ");
        }
    } while (valor < 0);
    return valor;
}
```

Função “ajustarIDs”: Ao excluir um produto, essa função irá ajustar os IDs dos outros produtos listados, para que assim não fique nenhum “buraco” entre eles. Já que o ID não é mais entrado pelo usuário, o próprio programa faz isso.

```c
// Função para ajustar IDs dos produtos
void ajustarIDs(Produto* produtos[], int* totalProdutos) {
    for (int i = 0; i < *totalProdutos; i++) {
        produtos[i]->ID = i + 1;
    }
}
```

Função “alterarQuantidade”: Seu objetivo é alterar apenas o valor da quantidade de um produto encontrado pelo ID. Seria para quando chegasse mais produtos na loja e o funcionário precisasse alterar a quantidade de produtos que estão no estoque.

```c
// Função para alterar apenas a quantidade em estoque
void alterarQuantidade(Produto* produtos[], int totalProdutos) {
    int id, novaQuantidade;
    bool encontrado = false;

    printf("Digite o ID do produto que deseja alterar a quantidade: ");
    id = lerInt();

    for (int i = 0; i < totalProdutos; i++) {
        if (produtos[i]->ID == id) {
            printf("Digite a nova quantidade em estoque: ");
            novaQuantidade = lerIntPositivo();
            produtos[i]->qntdEstoque = novaQuantidade;
            printf("Quantidade alterada com sucesso!\n");
            encontrado = true;
            break;
        }
    }
    if (!encontrado) {
        printf("Produto com ID %d não encontrado.\n", id);
    }
}
```

Função “VenderProduto”: Essa função irá facilitar a venda dos produtos para a loja, pois é necessário somente o ID do produto e a quantidade que será vendida. Ao realizar a venda, a quantidade vendida é subtraída do estoque e o valor final da venda aparece no terminal.

```c
// Função para vender um produto
void venderProduto(Produto* produtos[], int totalProdutos) {
    int id, quantidadeVendida;
    bool encontrado = false;

    printf("Digite o ID do produto que deseja vender: ");
    id = lerInt();
    printf("Digite a quantidade a ser vendida: ");
    quantidadeVendida = lerIntPositivo();

    for (int i = 0; i < totalProdutos; i++) {
        if (produtos[i]->ID == id) {
            if (produtos[i]->qntdEstoque >= quantidadeVendida) {
                double valorVenda = quantidadeVendida * produtos[i]->valorProduto;
                produtos[i]->qntdEstoque -= quantidadeVendida;
                printf("Venda realizada com sucesso! Valor da venda: %.2f\n", valorVenda);
            } else {
                printf("Quantidade insuficiente em estoque para realizar a venda.\n");
            }
            encontrado = true;
            break;
        }
    }
    if (!encontrado) {
        printf("Produto com ID %d não encontrado.\n", id);
    }
}
```

Função “aumentarPreco”: Ela funciona como o contrário da função de desconto, já que aumenta o preço do produto, encontrado por ID, de acordo com a porcentagem digitada.

```c
// Função para aumentar o preço dos produtos
void aumentarPreco(Produto* produtos[], int totalProdutos) {
    int id;
    double aumento;
    bool encontrado = false;

    printf("Digite o ID do produto para aumentar o preço: ");
    id = lerInt();
    printf("Digite o percentual de aumento (exemplo, para 10%%, digite 10): ");
    aumento = lerDoublePositivo();

    for (int i = 0; i < totalProdutos; i++) {
        if (produtos[i]->ID == id) {
            produtos[i]->valorProduto *= (1 + aumento / 100.0);
            printf("Aumento aplicado com sucesso! Novo valor: %.2f\n", produtos[i]->valorProduto);
            encontrado = true;
            break;
        }
    }
    if (!encontrado) {
        printf("Produto com ID %d não encontrado.\n", id);
    }
}
```

3.3.2. Alterações realizadas em funções já existentes

As imagens que estão com destaques em vermelho são do código original enviado por e-mail, já aquelas que estão com destaque em verde, são do código refatorado. Além disso, cada função estará em uma página separada, já que ocupam muito espaço.

Função “SetProduto”: Agora o ID não é mais escolhido pelo usuário, o que permite a organização dos IDs depois de uma exclusão, e garante que eles serão únicos, ou seja, não é possível existir dois produtos com o mesmo ID.

```
14- // Função para criar um produto
15- Produto* SetProduto(int ID, char* nomeProd, int qntdEstoque, double valorProduto) {
16      Produto* prod = (Produto*)malloc(sizeof(Produto));
17      if (prod != NULL) {
18-         prod->ID = ID;
19          strcpy(prod->nomeProd, nomeProd);
20          prod->qntdEstoque = qntdEstoque;
21          prod->valorProduto = valorProduto;
22      }
23      return prod;
24  }
```

```
14+ // Função para criar um produto com ID automático
15+ Produto* SetProdutoAutoID(int* totalProdutos, char* nomeProd, int qntdEstoque, double valorProduto) {
16      Produto* prod = (Produto*)malloc(sizeof(Produto));
17      if (prod != NULL) {
18+         prod->ID = (*totalProdutos) + 1;
19          strcpy(prod->nomeProd, nomeProd);
20          prod->qntdEstoque = qntdEstoque;
21          prod->valorProduto = valorProduto;
22      }
23      return prod;
24  }
```

Função “incluirProduto”: Todas as entradas de dados não são mais realizadas diretamente nessa função, eles são redirecionados para as funções novas criadas somente para esse objetivo. Além disso, a parte de ID não existe mais, já que agora ele se auto incrementa.

```
38  // Função para incluir um novo produto
39  void incluirProduto(Produto* produtos[], int* totalProdutos) {
40-     int id, quantidade;
41      double preco;
42      char nome[50];
43
44-     printf("Digite o ID do produto: ");
45-     scanf("%d", &id);
46      printf("Digite o nome do produto: ");
47-     scanf(" %[^\n]s", nome); // lê até o enter
48      printf("Digite a quantidade em estoque: ");
49-     scanf("%d", &quantidade);
50      printf("Digite o valor do produto: ");
51-     scanf("%lf", &preco);
52
53-     produtos[*totalProdutos] = SetProduto(id, nome, quantidade, preco);
54      (*totalProdutos)++;
55      printf("Produto adicionado com sucesso!\n");
56  }
```

```
108 // Função para incluir um novo produto
109 void incluirProduto(Produto* produtos[], int* totalProdutos) {
110+     int quantidade;
111     double preco;
112     char nome[50];
113
114     printf("Digite o nome do produto: ");
115+     lerString(nome, 50);
116     printf("Digite a quantidade em estoque: ");
117+     quantidade = lerIntPositivo();
118     printf("Digite o valor do produto: ");
119+     preco = lerDoublePositivo();
120
121+     produtos[*totalProdutos] = SetProdutoAutoID(totalProdutos, nome, quantidade, preco);
122     (*totalProdutos)++;
123     printf("Produto adicionado com sucesso!\n");
124 }
```

Função “alterarProduto”: Os processos de entrada de dados não estão mais sendo realizados nas funções em si, assim como em “incluirProduto”, são redirecionados para as novas funções com esses objetivos

```
58  // Função para alterar um produto existente por ID
59  void alterarProduto(Produto* produtos[], int totalProdutos) {
60      int id, novaQuantidade;
61      double novoPreco;
62      char novoNome[50];
63      bool encontrado = false;
64
65      printf("Digite o ID do produto que deseja alterar: ");
66-     scanf("%d", &id);
67
68      for (int i = 0; i < totalProdutos; i++) {
69          if (produtos[i]->ID == id) {
70              printf("Digite o novo nome do produto: ");
71-             scanf(" %[^\n]s", novoNome);
72              printf("Digite a nova quantidade em estoque: ");
73-             scanf("%d", &novaQuantidade);
74              printf("Digite o novo valor do produto: ");
75-             scanf("%lf", &novoPreco);
76
77              strcpy(produtos[i]->nomeProd, novoNome);
78              produtos[i]->qntdEstoque = novaQuantidade;
79              produtos[i]->valorProduto = novoPreco;
80              printf("Produto alterado com sucesso!\n");
81              encontrado = true;
82              break;
83          }
84      }
85      if (!encontrado) {
86          printf("Produto com ID %d não encontrado.\n", id);
87      }
88  }
```

```
133 // Função para alterar um produto existente por ID
134 void alterarProduto(Produto* produtos[], int totalProdutos) {
135     int id, novaQuantidade;
136     double novoPreco;
137     char novoNome[50];
138     bool encontrado = false;
139
140     printf("Digite o ID do produto que deseja alterar: ");
141+    id = lerInt();
142
143     for (int i = 0; i < totalProdutos; i++) {
144         if (produtos[i]->ID == id) {
145             printf("Digite o novo nome do produto: ");
146+            lerString(novoNome, 50);
147             printf("Digite a nova quantidade em estoque: ");
148+            novaQuantidade = lerIntPositivo();
149             printf("Digite o novo valor do produto: ");
150+            novoPreco = lerDoublePositivo();
151
152             strcpy(produtos[i]->nomeProd, novoNome);
153             produtos[i]->qntdEstoque = novaQuantidade;
154             produtos[i]->valorProduto = novoPreco;
155             printf("Produto alterado com sucesso!\n");
156             encontrado = true;
157             break;
158         }
159     }
160     if (!encontrado) {
161         printf("Produto com ID %d não encontrado.\n", id);
162     }
163 }
```

Função “consultarProduto”: As entradas de dados também foram redirecionadas para as novas funções.

```
90  // Função para consultar um produto por ID
91  void consultarProduto(Produto* produtos[], int totalProdutos) {
92      int id;
93      bool encontrado = false;
94
95      printf("Digite o ID do produto que deseja consultar: ");
96-     scanf("%d", &id);
97
98      for (int i = 0; i < totalProdutos; i++) {
99          if (produtos[i]->ID == id) {
100             imprimeProduto(produtos[i]);
101             encontrado = true;
102             break;
103         }
104     }
105     if (!encontrado) {
106         printf("Produto com ID %d não encontrado.\n", id);
107     }
108 }
```

```
188 // Função para consultar um produto por ID
189 void consultarProduto(Produto* produtos[], int totalProdutos) {
190     int id;
191     bool encontrado = false;
192
193     printf("Digite o ID do produto que deseja consultar: ");
194+    id = lerInt();
195
196     for (int i = 0; i < totalProdutos; i++) {
197         if (produtos[i]->ID == id) {
198             imprimeProduto(produtos[i]);
199             encontrado = true;
200             break;
201         }
202     }
203     if (!encontrado) {
204         printf("Produto com ID %d não encontrado.\n", id);
205     }
206 }
```

Função “excluirProduto”: A entrada de dados foi redirecionada para as novas funções e, agora, a função de exclusão utiliza a função “ajustarIDs”, para reorganizar todos os IDs dos produtos, evitando buracos que podem ser criados nessas exclusões.

```
110 // Função para excluir um produto por ID
111 void excluirProduto(Produto* produtos[], int* totalProdutos) {
112     int id;
113     bool encontrado = false;
114 
115     printf("Digite o ID do produto que deseja excluir: ");
116-    scanf("%d", &id);
117 
118     for (int i = 0; i < *totalProdutos; i++) {
119         if (produtos[i]->ID == id) {
120             free(produtos[i]); // libera memória do produto
121             for (int j = i; j < *totalProdutos - 1; j++) {
122                 produtos[j] = produtos[j + 1];
123             }
124             (*totalProdutos)--;
125             printf("Produto excluído com sucesso!\n");
126             encontrado = true;
127             break;
128         }
129     }
130     if (!encontrado) {
131         printf("Produto com ID %d não encontrado.\n", id);
132     }
133 }
```

```
208 // Função para excluir um produto por ID
209 void excluirProduto(Produto* produtos[], int* totalProdutos) {
210     int id;
211     bool encontrado = false;
212 
213     printf("Digite o ID do produto que deseja excluir: ");
214+    id = lerInt();
215 
216     for (int i = 0; i < *totalProdutos; i++) {
217         if (produtos[i]->ID == id) {
218             free(produtos[i]); // libera memória do produto
219             for (int j = i; j < *totalProdutos - 1; j++) {
220                 produtos[j] = produtos[j + 1];
221             }
222             (*totalProdutos)--;
223+            ajustarIDs(produtos, totalProdutos); // Ajustar IDs após exclusão
224             printf("Produto excluído com sucesso!\n");
225             encontrado = true;
226             break;
227         }
228     }
229     if (!encontrado) {
230         printf("Produto com ID %d não encontrado.\n", id);
231     }
232 }
```

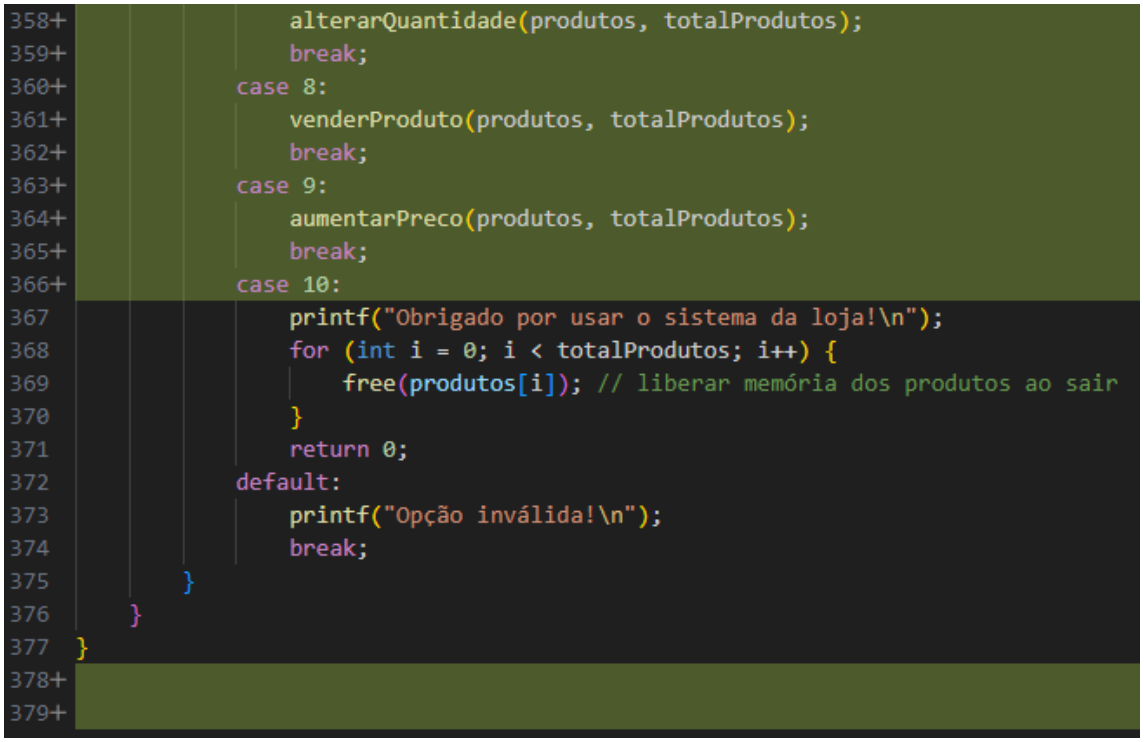
Função “aplicarDesconto”: A entrada de dados foi redirecionada para as novas funções e novos delimitadores foram incluídos, para que a quantidade de desconto seja no mínimo 0% e no máximo 100%.

```
146 // Função para aplicar desconto ao produto
147 void aplicarDesconto(Produto* produtos[], int totalProdutos) {
148     int id;
149     double desconto;
150     bool encontrado = false;
151 
152     printf("Digite o ID do produto para aplicar o desconto: ");
153-    scanf("%d", &id);
154-    printf("Digite o percentual de desconto (exemplo, para 10%%, digite 10): ");
155-    scanf("%lf", &desconto);
156 
157     for (int i = 0; i < totalProdutos; i++) {
158         if (produtos[i]->ID == id) {
159             produtos[i]->valorProduto *= (1 - desconto / 100.0);
160             printf("Desconto aplicado com sucesso! Novo valor: %.2f\n", produtos[i]->valorProduto);
161             encontrado = true;
162             break;
163         }
164     }
165     if (!encontrado) {
166         printf("Produto com ID %d não encontrado.\n", id);
167     }
168 }
```

```
245 // Função para aplicar desconto ao produto
246 void aplicarDesconto(Produto* produtos[], int totalProdutos) {
247     int id;
248     double desconto;
249     bool encontrado = false;
250 
251     printf("Digite o ID do produto para aplicar o desconto: ");
252+    id = lerInt();
253+    printf("Digite o percentual de desconto (0 a 100): ");
254+    do {
255+        desconto = lerDouble();
256+        if (desconto < 0 || desconto > 100) {
257+            printf("Percentual de desconto inválido. Digite um valor entre 0 e 100: ");
258+        }
259+    } while (desconto < 0 || desconto > 100);
260 
261     for (int i = 0; i < totalProdutos; i++) {
262         if (produtos[i]->ID == id) {
263             produtos[i]->valorProduto *= (1 - desconto / 100.0);
264             printf("Desconto aplicado com sucesso! Novo valor: %.2f\n", produtos[i]->valorProduto);
265             encontrado = true;
266             break;
267         }
268     }
269     if (!encontrado) {
270         printf("Produto com ID %d não encontrado.\n", id);
271     }
272 }
```

3.3.3. Alterações realizadas no “int main()”

Nessa parte do código, apenas foram adicionados mais opções no menu, para corresponder ao aumento de funções, e mais casos no “switch case” para realizar essas novas funções.



```
358+                alterarQuantidade(produtos, totalProdutos);
359+                break;
360+            case 8:
361+                venderProduto(produtos, totalProdutos);
362+                break;
363+            case 9:
364+                aumentarPreco(produtos, totalProdutos);
365+                break;
366+            case 10:
367                 printf("Obrigado por usar o sistema da loja!\n");
368                 for (int i = 0; i < totalProdutos; i++) {
369                     free(produtos[i]); // liberar memória dos produtos ao sair
370                 }
371                 return 0;
372             default:
373                 printf("Opção inválida!\n");
374                 break;
375         }
376     }
377 }
378+
379+
```

4. Conclusão

O código inicial atendia bem os requisitos propostos pela atividade e cumpria bem o seu papel, além de estar bem modularizado, faltando apenas funções que trabalham somente com I/O. Entretanto, ainda existiam alguns pontos que precisavam de melhoria, que foram as mudanças realizadas (todas estão presentes no código refatorado).

O primeiro programa apresentava alguns erros que deveriam ser deduzidos pelo próprio programador e não necessáriamente dito pelos contratantes. Um exemplo seria a possibilidade de digitar letras ou símbolos onde deveria ser somente números, o que acabava “quebrando” o código, além de permitir criar um desconto negativo, aumentando o preço do produto, ou maior do que 100%, o que deixava o produto com preço negativo.

Mas, no geral, o código atendia a todos os requisitos pedidos formalmente. Já o código refatorado, que está presente no github, incorporou as melhorias necessárias, corrigindo os erros mencionados e adicionando novas funcionalidades para ampliar as capacidades do programa.