**Autor: Matheus Arthur Moreira da Silva**
**Data: 29/11/2024**
**Professor Carlos Verissimo**
**Programação Estruturada e Modular**

**Análise crítica do código 23**

**Código 23:**

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

#define MAX_PRODUTOS 100

// Estrutura para armazenar informações de um produto
typedef struct {
    int id;
    char nome[50];
    int quantidadeEmEstoque;
    double valorDoProduto;
} Produto;

// Declaração das funções CRUD
void incluirProduto(Produto *produtos, int *quantidade);
Produto* consultarProduto(Produto *produtos, int quantidade, int id);
void alterarProduto(Produto *produtos, int quantidade);
void excluirProduto(Produto *produtos, int *quantidade);
void imprimirProduto(Produto *produto);
void venderProduto(Produto *produtos, int quantidade);

// Função principal
int main() {
    Produto produtos[MAX_PRODUTOS];
    int quantidade = 0;
    int opcao;

    do {
        printf("\nMenu:\n");
        printf("1 - Incluir produto\n");
        printf("2 - Consultar produto\n");
        printf("3 - Alterar produto\n");
        printf("4 - Excluir produto\n");
        printf("5 - Vender produto\n");
        printf("0 - Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);
```

```
        switch (opcao) {
            case 1:
                incluirProduto(produtos, &quantidade);
                break;
            case 2: {
                int id;
                printf("Digite o ID do produto a ser consultado: ");
                scanf("%d", &id);
                Produto *produto = consultarProduto(produtos, quantidade, id);
                if (produto) {
                    imprimirProduto(produto);
                } else {
                    printf("Produto não encontrado.\n");
                }
                break;
            }
            case 3:
                alterarProduto(produtos, quantidade);
                break;
            case 4:
                excluirProduto(produtos, &quantidade);
                break;
            case 5:
                venderProduto(produtos, quantidade);
                break;
            case 0:
                printf("Saindo...\n");
                break;
            default:
                printf("Opção inválida.\n");
                break;
        }
    } while (opcao != 0);

    return 0;
}

// Função para incluir um novo produto
void incluirProduto(Produto *produtos, int *quantidade) {
    if (*quantidade < MAX_PRODUTOS) {
        Produto *produto = &produtos[*quantidade];
        printf("Digite o ID do produto: ");
        scanf("%d", &produto->id);
        printf("Digite o nome do produto: ");
        scanf(" %[^\n]", produto->nome);
        printf("Digite a quantidade em estoque: ");
```

```
        scanf("%d", &produto->quantidadeEmEstoque);
        printf("Digite o valor do produto: ");
        scanf("%lf", &produto->valorDoProduto);

        (*quantidade)++;
        printf("Produto incluído com sucesso!\n");
    } else {
        printf("Capacidade máxima de produtos atingida.\n");
    }
}

// Função para consultar um produto pelo ID
Produto* consultarProduto(Produto *produtos, int quantidade, int id) {
    for (int i = 0; i < quantidade; i++) {
        if (produtos[i].id == id) {
            return &produtos[i];
        }
    }
    return NULL;
}

// Função para alterar um produto pelo ID
void alterarProduto(Produto *produtos, int quantidade) {
    int id;
    printf("Digite o ID do produto a ser alterado: ");
    scanf("%d", &id);

    Produto *produto = consultarProduto(produtos, quantidade, id);
    if (produto) {
        printf("Digite o novo nome do produto: ");
        scanf(" %[^\n]", produto->nome);
        printf("Digite a nova quantidade em estoque: ");
        scanf("%d", &produto->quantidadeEmEstoque);
        printf("Digite o novo valor do produto: ");
        scanf("%lf", &produto->valorDoProduto);
        printf("Produto alterado com sucesso!\n");
    } else {
        printf("Produto não encontrado.\n");
    }
}

// Função para excluir um produto pelo ID
void excluirProduto(Produto *produtos, int *quantidade) {
    int id;
    printf("Digite o ID do produto a ser excluído: ");
    scanf("%d", &id);
```

```
    for (int i = 0; i < *quantidade; i++) {
        if (produtos[i].id == id) {
            for (int j = i; j < *quantidade - 1; j++) {
                produtos[j] = produtos[j + 1];
            }
            (*quantidade)--;
            printf("Produto excluído com sucesso!\n");
            return;
        }
    }
    printf("Produto não encontrado.\n");
}

// Função para imprimir os dados de um produto
void imprimirProduto(Produto *produto) {
    printf("ID: %d\n", produto->id);
    printf("Nome: %s\n", produto->nome);
    printf("Quantidade em Estoque: %d\n", produto->quantidadeEmEstoque);
    printf("Valor do Produto: %.2f\n", produto->valorDoProduto);
}

// Função para vender um produto, reduzindo a quantidade em estoque
void venderProduto(Produto *produtos, int quantidade) {
    int id, quantidadeVendida;
    printf("Digite o ID do produto a ser vendido: ");
    scanf("%d", &id);

    Produto *produto = consultarProduto(produtos, quantidade, id);
    if (produto) {
        printf("Digite a quantidade a ser vendida: ");
        scanf("%d", &quantidadeVendida);

        if (quantidadeVendida <= produto->quantidadeEmEstoque) {
            produto->quantidadeEmEstoque -= quantidadeVendida;
            printf("Venda realizada com sucesso!\n");
            imprimirProduto(produto);
        } else {
            printf("Quantidade em estoque insuficiente.\n");
        }
    } else {
        printf("Produto não encontrado.\n");
    }
}
```

**Início da análise**

O código inicia com a declaração de bibliotecas que serão usadas no programa.

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

Após isso, ele declara uma variável global com o uso do comando “#define”.

```c
#define MAX_PRODUTOS 100
```

É criada a struct que armazena informações dos produtos, é comentada, e as variáveis estão com os nomes no modelo camelCase, porém apresenta Hardcode em “char nome[50]”.

```c
// Estrutura para armazenar informações de um produto
typedef struct {
    int id;
    char nome[50];
    int quantidadeEmEstoque;
    double valorDoProduto;
} Produto;
```

Declaração de variáveis CRUD, que também estão em camelCase.

```c
// Declaração das funções CRUD
void incluirProduto(Produto *produtos, int *quantidade);
Produto* consultarProduto(Produto *produtos, int quantidade, int id);
void alterarProduto(Produto *produtos, int quantidade);
void excluirProduto(Produto *produtos, int *quantidade);
void imprimirProduto(Produto *produto);
void venderProduto(Produto *produtos, int quantidade);
```

Todas as funções possuem nomes bem claros e intuitivos, que fazem com que seja fácil de saber a função de cada uma das funções.

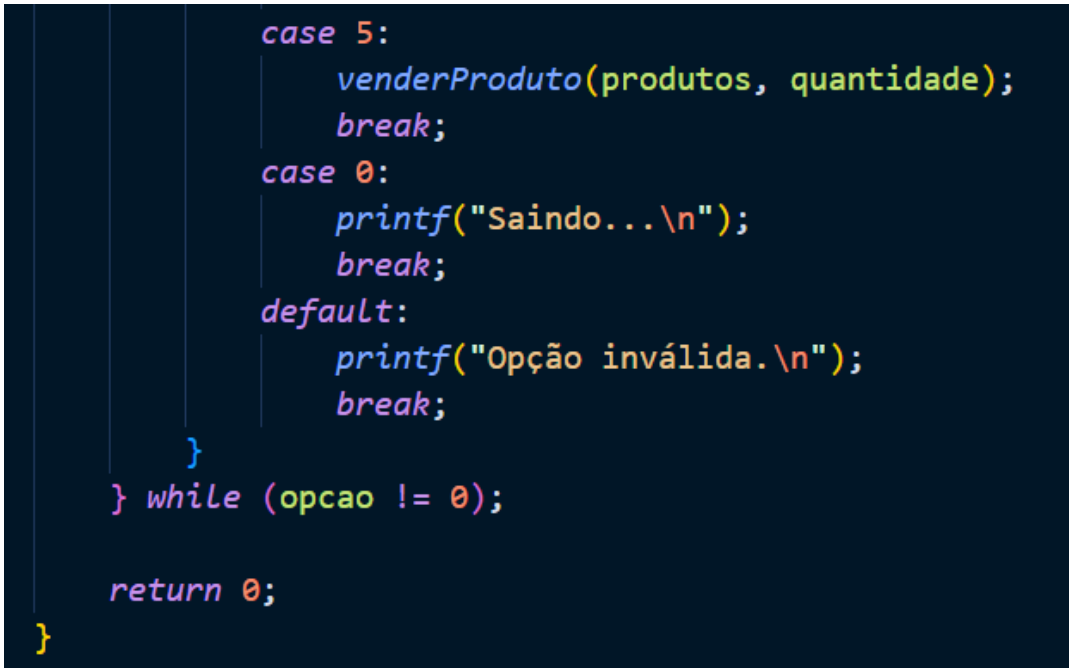
Aqui ele apenas faz menção a todas estas funções, ainda não detalhou o que cada uma delas faz.

A partir da linha 24, ele inicia o “int main” do programa.

```c
// Função principal
int main() {
    Produto produtos[MAX_PRODUTOS];
    int quantidade = 0;
    int opcao;

    do {
        printf("\nMenu:\n");
        printf("1 - Incluir produto\n");
        printf("2 - Consultar produto\n");
        printf("3 - Alterar produto\n");
        printf("4 - Excluir produto\n");
        printf("5 - Vender produto\n");
        printf("0 - Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);
```

```c
switch (opcao) {
    case 1:
        incluirProduto(produtos, &quantidade);
        break;
    case 2: {
        int id;
        printf("Digite o ID do produto a ser consultado: ");
        scanf("%d", &id);
        Produto *produto = consultarProduto(produtos, quantidade, id);
        if (produto) {
            imprimirProduto(produto);
        } else {
            printf("Produto não encontrado.\n");
        }
        break;
    }
    case 3:
        alterarProduto(produtos, quantidade);
        break;
    case 4:
        excluirProduto(produtos, &quantidade);
        break;
```



```
            case 5:
                venderProduto(produtos, quantidade);
                break;
            case 0:
                printf("Saindo...\n");
                break;
            default:
                printf("Opção inválida.\n");
                break;
        }
    } while (opcao != 0);

    return 0;
}
```

Ele inicia o código criando o array que irá armazenar os produtos:
“Produto produtos[MAX_PRODUTOS];”, “Produto” é a struct, “produtos” é o nome do array, “MAX_PRODUTOS” é a variável global que nesse código tem valor 100.

Após isso ele cria 2 variáveis int (quantidade e opcao), a variável “quantidade” é iniciada com valor 0 e ela controlará os elementos preenchidos na array, já a variável opcao é usada para armazenar a escolha do usuário no menu.

É criada a estrutura do menu com ajuda da função “do while”, onde ele realiza comandos “printf” que irão guiar o usuário do programa e garante que o usuário irá visualizar o menu ao menos 1 vez.

A linha “scanf(“%d”, &opcao);” permite que o usuário digite qual a opção ele deseja selecionar para usar, por exemplo “3 – Alterar produto”.

O criador do código fez uso de “switch case” para dividir as funções em casos específicos que foram selecionados pelo usuário acima, nestes casos, ele usa as funções que serão “montadas” logo após o fim do int main.

Para finalizar o main, ele fecha o comando “do while” com “while (opcao != 0);”, pois desta forma quando o usuário selecionar a opção “0 – Sair”, isso irá fazer com que o while pare e finalize o programa. Além de ter o return 0.

Depois de finalizar o int main, começa o código das funções.

A primeira função é “incluirProduto”:

```c
// Função para incluir um novo produto
void incluirProduto(Produto *produtos, int *quantidade) {
    if (*quantidade < MAX_PRODUTOS) {
        Produto *produto = &produtos[*quantidade];
        printf("Digite o ID do produto: ");
        scanf("%d", &produto->id);
        printf("Digite o nome do produto: ");
        scanf(" %[^\n]", produto->nome);
        printf("Digite a quantidade em estoque: ");
        scanf("%d", &produto->quantidadeEmEstoque);
        printf("Digite o valor do produto: ");
        scanf("%lf", &produto->valorDoProduto);

        (*quantidade)++;
        printf("Produto incluído com sucesso!\n");
    } else {
        printf("Capacidade máxima de produtos atingida.\n");
    }
}
```

A primeira coisa feita no código é verificar se o número de produtos cadastrados é menor que o limite definido pela variável global. Depois ele tem uma variável chamada “produto” que aponta para o próximo elemento disponível no array “produtos”, e o índice “*quantidade” indica a posição atual que está vazia.
Imprime na tela os pedidos de dados com “printf” e recebe com “scanf”. “(*quantidade)++;” aumenta em 1 a quantidade de produtos armazenados.

A segunda função é “consultarProduto”:

```c
// Função para consultar um produto pelo ID
Produto* consultarProduto(Produto *produtos, int quantidade, int id) {
    for (int i = 0; i < quantidade; i++) {
        if (produtos[i].id == id) {
            return &produtos[i];
        }
    }
    return NULL;
}
```

O código faz um loop “for” para percorrer o array. O índice ‘i’ vai de 0 até (quantidade -1), garantindo que todos os produtos sejam verificados. Dentro do loop, a função compara o ID do produto atual “produtos[i].id” com o ID fornecido como parâmetro “id”. Se for encontrada uma correspondência, a função retorna um ponteiro para o produto correspondente “&produtos[i]”, permitindo que ele seja acessado fora da função.

Se o loop não encontrar nenhum produto, a função retorna “NULL”, que indica que não existe produto no array.
A terceira função é “alterarProduto”:

```c
// Função para alterar um produto pelo ID
void alterarProduto(Produto *produtos, int quantidade) {
    int id;
    printf("Digite o ID do produto a ser alterado: ");
    scanf("%d", &id);

    Produto *produto = consultarProduto(produtos, quantidade, id);
    if (produto) {
        printf("Digite o novo nome do produto: ");
        scanf(" %[^\n]", produto->nome);
        printf("Digite a nova quantidade em estoque: ");
        scanf("%d", &produto->quantidadeEmEstoque);
        printf("Digite o novo valor do produto: ");
        scanf("%lf", &produto->valorDoProduto);
        printf("Produto alterado com sucesso!\n");
    } else {
        printf("Produto não encontrado.\n");
    }
}
```

Essa função permite que o usuário altere as informações de um produto específico, identificando-o pelo ID. Ele solicita o ID do produto que deseja alterar e o valor digitado é armazenado na variável “id”.

Ele chama a função “consultarProduto” para localizar o produto com o ID fornecido, se for encontrado, a variável “produto” irá armazená-lo.

Recebe o ID do produto a ser alterado, busca o produto no array usando “consultarProduto”, se encontrado, atualiza os campos “nome”, “quantidadeEmEstoque” e “valorDoProduto”.

A quarta função é “excluirProduto”:

```c
// Função para excluir um produto pelo ID
void excluirProduto(Produto *produtos, int *quantidade) {
    int id;
    printf("Digite o ID do produto a ser excluído: ");
    scanf("%d", &id);

    for (int i = 0; i < *quantidade; i++) {
        if (produtos[i].id == id) {
            for (int j = i; j < *quantidade - 1; j++) {
                produtos[j] = produtos[j + 1];
            }
            (*quantidade)--;
            printf("Produto excluído com sucesso!\n");
            return;
        }
    }
    printf("Produto não encontrado.\n");
}
```

A função solicita ao usuário o ID do produto a ser excluído e percorre o array de produtos para localizar o produto com o ID fornecido.

Se encontrado, remove o produto deslocando os itens seguintes no array e atualiza a quantidade total de produtos, além de exibir uma mensagem de sucesso. Se não for encontrado, exibe uma mensagem indicando que o produto não existe.

Um loop “for” percorre o array de produtos para encontrar o produto com o ID fornecido. A comparação “produtos[i].id == id” verifica se o produto na posição ‘i’ corresponde ao ID buscado.

A quinta função é “imprimirProduto”:

```c
// Função para imprimir os dados de um produto
void imprimirProduto(Produto *produto) {
    printf("ID: %d\n", produto->id);
    printf("Nome: %s\n", produto->nome);
    printf("Quantidade em Estoque: %d\n", produto->quantidadeEmEstoque);
    printf("Valor do Produto: %.2f\n", produto->valorDoProduto);
}
```

A função acessa os campos da struct “Produto” usando o operador “->”, que é utilizado para acessar membros de structs por meio de ponteiros.

Essa função não é muito utilizada, sendo útil apenas no case 2 e na próxima função.

A sexta e última função é “venderProduto”:

```c
// Função para vender um produto, reduzindo a quantidade em estoque
void venderProduto(Produto *produtos, int quantidade) {
    int id, quantidadeVendida;
    printf("Digite o ID do produto a ser vendido: ");
    scanf("%d", &id);

    Produto *produto = consultarProduto(produtos, quantidade, id);
    if (produto) {
        printf("Digite a quantidade a ser vendida: ");
        scanf("%d", &quantidadeVendida);

        if (quantidadeVendida <= produto->quantidadeEmEstoque) {
            produto->quantidadeEmEstoque -= quantidadeVendida;
            printf("Venda realizada com sucesso!\n");
            imprimirProduto(produto);
        } else {
            printf("Quantidade em estoque insuficiente.\n");
        }
    } else {
        printf("Produto não encontrado.\n");
    }
}
```

A função solicita o ID do produto, o usuário insere o ID do produto que deseja vender, depois ele busca o produto e usa a função “consultarProduto” para localizar o produto no array. Se o produto não for encontrado, exibe uma mensagem informando que ele não existe.

Também solicita a quantidade a ser vendida, verifica se o estoque é suficiente e reduz a quantidade em estoque e exibe os dados atualizados do produto com a função “imprimirProduto”.

**Análise específica de requisitos**

**Modularização**: A modularização do código está bem estruturada, com cada funcionalidade separada em funções específicas, como inclusão, consulta, alteração, exclusão e venda de produtos. Todas as operações são realizadas diretamente na struct “Produto” por meio de ponteiros, o que facilita a manipulação dos dados.

(Incluir produto, Consultar produto (por código), Alterar produto, Excluir produto, Vender produto) = CRUD

**CRUD**: Quando damos “run” no programa, vemos que as funções que aparecem no menu são os presentes no CRUD (Incluir, Consultar, Alterar, Excluir e Vender produtos). Elas estão bem funcionais, e com pequenos erros que serão solucionados na sua grande maioria no código refatorado.

**Ponteiros**: O uso de ponteiros no código é eficaz para as operações solicitadas, tornando o programa mais eficiente e direto.

**Struct**: O uso de structs é apropriado neste código, pois organiza e agrupa os dados de forma clara e eficiente. A implementação está simples e funcional, atendendo bem ao propósito do programa. Possui os requisitos necessários: “ID” (inteiro), “nome” (string), “QuantidadeEmEstoque” (int), “ValorDoProduto” (double).

**Funções**: As funções para incluir produto, alterar produto, consultar produto, excluir produto, vender produtos e imprimir os dados do produto estão todas funcionando, comentadas, camelCase e utilizando ponteiros.

**Programa Refatorado**

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

#define MAX_PRODUTOS 100
#define MAX_NOME 50

// Funções de validação de entrada
int lerInteiroPositivo() {
    int num;
    while (scanf("%d", &num) != 1 || num < 0) {
        while (getchar() != '\n'); // Limpa o buffer
        printf("Entrada inválida! Digite um valor válido: ");
    }
    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;
}

void lerString(char *str, int tamanho) {
    getchar();
    fgets(str, tamanho, stdin);
    str[strcspn(str, "\n")] = '\0'; // Remove o '\n' da string
}

int isStringValida(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
}
// Estrutura para armazenar informações de um produto
typedef struct {
    int id;
    char nome[MAX_NOME];
    int quantidadeEmEstoque;
    double valorDoProduto;
} Produto;

// Declaração das funções CRUD
void incluirProduto(Produto *produtos, int *quantidade);
Produto* consultarProduto(Produto *produtos, int quantidade, int id);
void alterarProduto(Produto *produtos, int quantidade);
void excluirProduto(Produto *produtos, int *quantidade);
void imprimirProduto(Produto *produto);
void venderProduto(Produto *produtos, int quantidade);
void aplicarDesconto(Produto *produtos, int quantidade);

// Função principal
int main() {
    Produto produtos[MAX_PRODUTOS];
    int quantidade = 0;
    int opcao;

    do {
        printf("\nMenu:\n");
        printf("1 - Incluir produto\n");
        printf("2 - Consultar produto\n");
        printf("3 - Alterar produto\n");
        printf("4 - Excluir produto\n");
        printf("5 - Vender produto\n");
        printf("6 - Aplicar desconto\n");
        printf("0 - Sair\n");
        printf("Escolha uma opção: ");
        opcao = lerInteiroPositivo();
        switch (opcao) {
            case 1:
                incluirProduto(produtos, &quantidade);
                break;
            case 2: {
                int id;
```

```
            printf("Digite o ID do produto a ser consultado: ");
            id = lerInteiroPositivo();
            Produto *produto = consultarProduto(produtos, quantidade, id);
            if (produto) {
                imprimirProduto(produto);
            } else {
                printf("Produto não encontrado.\n");
            }
            break;
        }
        case 3:
            alterarProduto(produtos, quantidade);
            break;
        case 4:
            excluirProduto(produtos, &quantidade);
            break;
        case 5:
            venderProduto(produtos, quantidade);
            break;
        case 6:
            aplicarDesconto(produtos, quantidade);
            break;
        case 0:
            printf("Saindo...\n");
            break;
        default:
            printf("Opção inválida.\n");
            break;
    }
} while (opcao != 0);

return 0;
}

// Função para incluir um novo produto
void incluirProduto(Produto *produtos, int *quantidade) {
    if (*quantidade < MAX_PRODUTOS) {
        int novoId;
        do {
            printf("Digite o ID do produto: ");
            novoId = lerInteiroPositivo();

            // Verificar se o ID já existe
            if (consultarProduto(produtos, *quantidade, novoId) != NULL) {
                printf("ID já existente. Escolha um ID diferente.\n");
            } else {
                break;
```

```
            }
        } while (1);

        Produto *produto = &produtos[*quantidade];
        produto->id = novoId;

        do {
            printf("Digite o nome do produto: ");
            lerString(produto->nome, MAX_NOME);

            if (!isStringValida(produto->nome)) {
                printf("Nome inválido! Use apenas letras e espaços.\n");
            }
        } while (!isStringValida(produto->nome));

        printf("Digite a quantidade em estoque: ");
        produto->quantidadeEmEstoque = lerInteiroPositivo();

        printf("Digite o valor do produto (Use ponto no lugar de vírgula): ");
        produto->valorDoProduto = lerDoublePositivo();

        (*quantidade)++;
        printf("Produto incluído com sucesso!\n");
    } else {
        printf("Capacidade máxima de produtos atingida.\n");
    }
}

// Função para consultar um produto pelo ID
Produto* consultarProduto(Produto *produtos, int quantidade, int id) {
    for (int i = 0; i < quantidade; i++) {
        if (produtos[i].id == id) {
            return &produtos[i];
        }
    }
    return NULL;
}

// Função para alterar um produto pelo ID
void alterarProduto(Produto *produtos, int quantidade) {
    int id;
    printf("Digite o ID do produto a ser alterado: ");
    id = lerInteiroPositivo();

    Produto *produto = consultarProduto(produtos, quantidade, id);
    if (produto) {
        do {
```

```
            printf("Digite o novo nome do produto: ");
            lerString(produto->nome, MAX_NOME);

            if (!isStringValida(produto->nome)) {
                printf("Nome inválido! Use apenas letras e espaços.\n");
            }
        } while (!isStringValida(produto->nome));

        printf("Digite a nova quantidade em estoque: ");
        produto->quantidadeEmEstoque = lerInteiroPositivo();

        printf("Digite o novo valor do produto (Use \'. no lugar de vírgula): ");
        produto->valorDoProduto = lerDoublePositivo();

        printf("Produto alterado com sucesso!\n");
    } else {
        printf("Produto não encontrado.\n");
    }
}

// Função para excluir um produto pelo ID
void excluirProduto(Produto *produtos, int *quantidade) {
    int id;
    printf("Digite o ID do produto a ser excluído: ");
    id = lerInteiroPositivo();

    for (int i = 0; i < *quantidade; i++) {
        if (produtos[i].id == id) {
            for (int j = i; j < *quantidade - 1; j++) {
                produtos[j] = produtos[j + 1];
            }
            (*quantidade)--;
            printf("Produto excluído com sucesso!\n");
            return;
        }
    }
    printf("Produto não encontrado.\n");
}

// Função para imprimir os dados de um produto
void imprimirProduto(Produto *produto) {
    printf("ID: %d\n", produto->id);
    printf("Nome: %s\n", produto->nome);
    printf("Quantidade em Estoque: %d\n", produto->quantidadeEmEstoque);
    printf("Valor do Produto: %.2f\n", produto->valorDoProduto);
}
```

```
// Função para vender um produto, reduzindo a quantidade em estoque
void venderProduto(Produto *produtos, int quantidade) {
    int id, quantidadeVendida;
    printf("Digite o ID do produto a ser vendido: ");
    id = lerInteiroPositivo();

    Produto *produto = consultarProduto(produtos, quantidade, id);
    if (produto) {
        printf("Digite a quantidade a ser vendida: ");
        quantidadeVendida = lerInteiroPositivo();

        if (quantidadeVendida <= produto->quantidadeEmEstoque) {
            produto->quantidadeEmEstoque -= quantidadeVendida;
            printf("Venda realizada com sucesso!\n");
            imprimirProduto(produto);
        } else {
            printf("Quantidade em estoque insuficiente.\n");
        }
    } else {
        printf("Produto não encontrado.\n");
    }
}

// Função para aplicar desconto em um produto pelo ID
void aplicarDesconto(Produto *produtos, int quantidade){
    int id;
    double desconto;

    printf("Informe o ID do produto para aplicar desconto: ");
    id = lerInteiroPositivo();

    // Busca o produto pelo ID
    for (int i=0; i < quantidade; i++) {
        if (produtos[i].id == id) {
            printf("Informe o percentual de desconto (0-100): ");
            desconto = lerDoublePositivo();

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

            produtos[i].valorDoProduto *= (1 - desconto / 100); // Aplica o desconto
            printf("Desconto aplicado com sucesso! Novo preço: %.2lf\n",
produtos[i].valorDoProduto);
            return;
        }
```

```
    }
        printf("Produto inexistente.\n");
}
```

**Mudanças feitas no código refatorado**

**Adição de bibliotecas**

O código original incluía 3 bibliotecas, (stdio.h, stdlib.h, string.h), e no refatorado foi adicionada a biblioteca “ctype.h”.

**Criação de novas funções**

“lerInteiroPositivo”, “lerDoublePositivo”, “lerString”, “isStringValida”, “aplicarDesconto”.

Essas funções também têm nome intuitivo e fácil de entender

**Tratamento na entrada de valores**

No código original, era usado “scanf()” sem nenhuma verificação de valores, fazendo com que se pudesse digitar letras na área de valor do produto, por exemplo.
No código novo, as funções “lerInteiroPositivo”, “lerDoublePositivo”, “lerString” e “isStringValida” irão verificar se o que o usuário está passando dados válidos para cada um dos requisitos do código.

**Menu**

Adicionada a opção “Aplicar Desconto” no menu de opções, visto que isso não existia no código antigo.

**Switch Case**

Adicionado um caso 6 de aplicar desconto.

**Tratamento de ID**

No código antigo era possível criar vários itens com o mesmo ID, no código refatorado não é possível ter itens repetidos com o mesmo ID, e caso queira adicionar mais produtos em um ID, é possível usar a opção “Alterar Produto”.

**Tratamento de erros**

Implementadas funções de leitura que limpam o buffer de entrada, pedem nova entrada em caso de valor inválido e garantem a entrada de valores positivos, impossibilitando preços ou quantidades negativas.

**Variável Global**

Adicionado “MAX_NOME” para ser a variável global para tamanho de nome.

**Conclusão**

O código antigo era bem-feito, possuía muitas qualidades como a modularização ótima, uso de structs e ponteiros correto e as funções estavam seguindo corretamente o que se pedia, porém possuía alguns erros pequenos e outros maiores.

Estes erros foram, em sua maioria, resolvidos pelo código refatorado, entre eles destaco a ausência de uma função para aplicar desconto nos produtos, que era pedido pelo contratante, além da pouca verificação na entrada de dados. Com a adição de novas funções que fazem esse papel, além da implementação de novas variáveis e até mesmo bibliotecas, foi possível reduzir o número de erros de forma magistral.

Mesmo assim, o código é bom e não precisa de muito para entender o que ele nos demonstra, se for bem utilizado, ele pode funcionar mesmo sem nenhuma mudança, mas é nítido que com as melhorias, o código se torna um próprio “upgrade”.