# Documentação do Código - Introdução

## Título do Arquivo

Gerenciamento de Produtos e Vendas

## Descrição do Arquivo

Este documento apresenta o código inicial de um programa desenvolvido em linguagem C, voltado para a gestão de produtos e vendas. O sistema é capaz de cadastrar produtos, gerenciar o estoque, realizar vendas, emitir notas fiscais, e gerar relatórios com base em informações armazenadas.

## Objetivo Geral

Desenvolver uma aplicação funcional para o gerenciamento de produtos e vendas, incluindo funcionalidades como cadastro, consultas, atualização de informações, e geração de relatórios financeiros.

## Bibliotecas Utilizadas

O código inclui as seguintes bibliotecas padrão do C, cada uma com funcionalidades específicas:

1. <stdio.h>: Biblioteca padrão para entrada e saída de dados (ex.: funções como printf e scanf).

2. <stdlib.h>: Fornece funções para alocação de memória, conversões, e controle do programa (ex.: malloc, exit).

3. <string.h>: Contém funções para manipulação de strings (ex.: strcpy, strlen).

4. <time.h>: Utilizada para capturar o tempo do sistema e operações baseadas em data/hora.

5. <locale.h>: Permite a configuração de localidade para suporte a caracteres especiais (ex.: acentuação).

6. <conio.h>: Inclui funções específicas para controle do terminal, como clrscr() para limpar a tela e métodos para manipulação de entrada do usuário.

## Constantes Definidas

As seguintes constantes foram criadas para garantir clareza e padronização no código:

- TAMANHO\_NOME\_PRODUTO (50): Tamanho máximo permitido para o nome de um produto.

- TAMANHO\_MARCA (30): Tamanho máximo permitido para o nome da marca de um produto.

- MAX (100): Número máximo de itens que podem ser armazenados no sistema.

Esses valores são definidos como macros usando #define, para facilitar a reutilização e eventual alteração futura.

## Estruturas Criadas

### Produto

A estrutura Produto foi criada para representar os produtos cadastrados no sistema. Cada produto possui as seguintes variáveis associadas:

- int codigo: Código único para identificação do produto.

- char nome\_produto[TAMANHO\_NOME\_PRODUTO]: Nome do produto.

- char marca[TAMANHO\_MARCA]: Marca do produto.

- int estoque: Quantidade disponível no estoque.

- float custo: Custo unitário do produto.

- float venda: Preço de venda do produto.

- float lucro: Diferença entre o preço de venda e o custo.

Objetivo: Essa estrutura é usada para funções de cadastro, consulta, alteração de produtos e relatórios financeiros.

### Vendas

A estrutura Vendas foi criada para gerenciar as transações realizadas pelo sistema. Cada venda possui as seguintes variáveis associadas:

- int codigo\_produto: Código do produto vendido.

- char nome\_produto[TAMANHO\_NOME\_PRODUTO]: Nome do produto vendido.

- int qtd\_produto: Quantidade de itens vendidos.

- float venda: Valor unitário do produto no momento da venda.

- float valor\_total: Valor total da venda (quantidade × preço unitário).

- int tipo\_pagamento: Indica o método de pagamento utilizado (ex.: cartão, dinheiro, etc.).

Objetivo: Essa estrutura é usada para registrar e gerenciar informações sobre vendas realizadas, além de auxiliar na geração de notas fiscais.

## Resumo das Funcionalidades

- Cadastro de Produtos: Estrutura Produto será usada para armazenar e gerenciar informações.

- Gestão de Estoque: Monitoramento da quantidade disponível no estoque.

- Registro de Vendas: Estrutura Vendas será usada para registrar todas as transações e gerar relatórios.

- Geração de Relatórios: Possibilidade de calcular lucro e gerar relatórios financeiros baseados nas transações registradas.

## Próximos Passos

- Implementar funções para manipulação das estruturas Produto e Vendas.

- Criar a lógica para leitura e gravação de dados em arquivos.

- Desenvolver uma interface de interação com o usuário para facilitar o cadastro e consulta.

## Funções

Abaixo está a descrição e a finalidade de cada função declarada no código. Essas funções serão posteriormente definidas e implementadas ao longo do programa.

**Objetivo da Declaração**

Declarar as funções antes da função principal int main() permite que elas sejam implementadas em qualquer ordem no código, garantindo que suas chamadas sejam reconhecidas, mesmo antes da definição completa. Isso melhora a organização e a legibilidade do código.

**Lista de Funções e Suas Finalidades**

1. **void cadastrarProduto();**  
   Função responsável pelo cadastro de novos produtos no sistema. Solicita dados do usuário e os armazena na estrutura Produto.
2. **void abrirArquivoProdutos(FILE \*arquivo);**  
   Abre o arquivo responsável por armazenar as informações dos produtos cadastrados. Garante que o arquivo esteja pronto para leitura/escrita.
3. **void limparBuffer();**  
   Limpa o buffer do teclado, evitando que caracteres residuais interfiram na leitura de entradas do usuário.
4. **void sairComQualquerTecla();**  
   Aguarda o pressionamento de uma tecla para que o usuário possa sair ou continuar no sistema.
5. **void limparTela();**  
   Limpa a tela do terminal, utilizando comandos específicos da biblioteca <conio.h>.
6. **int obterInteiroPositivo(const char \*mensagem);**  
   Solicita ao usuário a entrada de um número inteiro positivo, validando o valor antes de retorná-lo.
7. **float obterFloatPositivo(const char \*mensagem);**  
   Solicita ao usuário a entrada de um número decimal positivo, realizando validação semelhante à função acima.
8. **void alterarProduto();**  
   Permite alterar as informações de um produto previamente cadastrado no sistema.
9. **int obterQuantidadeProduto(int estoque);**  
   Obtém a quantidade de um produto em estoque, garantindo que o valor seja válido.
10. **void registrarVenda(Vendas \*venda, int codigo, const char \*nome\_produto, int qtd\_produto, float venda\_valor);**  
    Registra as informações de uma venda, incluindo detalhes do produto e valores.
11. **void gerarVendas();**  
    Gerencia o processo de geração de vendas, organizando as transações no sistema.
12. **void venderProdutos();**  
    Permite ao usuário realizar a venda de produtos, atualizando os dados de estoque e registrando a transação.
13. **void exibirCupomHeader(FILE \*arquivo, const char \*data, const char \*hora, int numero\_nota);**  
    Exibe o cabeçalho do cupom fiscal, com informações como data, hora e número da nota fiscal.
14. **void exibirDetalhesProduto(FILE \*arquivo, Vendas \*venda);**  
    Mostra detalhes de cada produto incluído em uma venda específica.
15. **void exibirCupomFooter(FILE \*arquivo, int totalItens, float valorTotal, int tipo, const char \*digitos);**  
    Adiciona o rodapé ao cupom fiscal, exibindo total de itens, valor total da compra e método de pagamento.
16. **void exibirNotaFiscal(Vendas \*ptr\_vetor\_vendas, int tamanho, int tipo);**  
    Gera e exibe uma nota fiscal baseada nas vendas realizadas.
17. **void gerarChaveAcesso(char \*sequencia);**  
    Gera uma sequência de caracteres única para servir como chave de acesso à nota fiscal.
18. **void exibirProdutos();**  
    Exibe todos os produtos cadastrados no sistema, com seus detalhes.
19. **int compararCodigoProduto(int numero);**  
    Compara um código de produto fornecido pelo usuário com os produtos registrados no sistema.
20. **void creditos();**  
    Exibe informações de crédito aos desenvolvedores.
21. **void abrirArquivoCsv(FILE \*csv, char path[]);**  
    Abre um arquivo CSV específico, utilizando o caminho fornecido.
22. **void gerarRelatorioProdutos();**  
    Gera um relatório detalhado dos produtos cadastrados, incluindo informações de estoque e preços.
23. **void abrirArquivoVendas(FILE \*arquivo);**  
    Abre o arquivo responsável por armazenar os dados de vendas.
24. **void gerarRelatorioVenda();**  
    Gera um relatório com informações detalhadas sobre as vendas realizadas.
25. **void gerarRelatorios();**  
    Gerencia a geração de relatórios variados, unificando produtos e vendas.
26. **int ler\_numero\_nota(const char \*nome\_arquivo);**  
    Lê o número da última nota fiscal salva no arquivo.
27. **int incrementar\_nota(const char \*nome\_arquivo);**  
    Incrementa o número da nota fiscal, garantindo que cada venda tenha um número único.
28. **void consultarVendas();**  
    Permite ao usuário consultar as vendas realizadas, com base em critérios específicos.

**Documentação do Código: Função main()**

**Descrição Geral**

**A função `main()` é o ponto de entrada do programa. Ela apresenta um menu interativo para o usuário, permitindo realizar diversas operações relacionadas ao sistema de gerenciamento de produtos e vendas.**

**Detalhamento do Código**

**1. Configuração de Codificação:**

**```c  
setlocale(LC\_ALL, "Portuguese");  
`**`` **Essa função configura o programa para aceitar a codificação de caracteres no formato Latino Básico. Isso é essencial para o suporte ao idioma Português, garantindo que caracteres acentuados sejam exibidos e processados corretamente.**

**2. Definição da Variável opcao:**

**```c  
int opcao;  
```  
A variável `opcao` armazena a escolha do usuário no menu principal. O tipo `int` é utilizado porque as opções do menu são números inteiros.**

**3. Menu Interativo (Laço do-while):**

**O menu é exibido ao usuário dentro de um laço `do-while`, que garante a repetição do menu até que o usuário escolha sair (opção `10`).**

**```c  
do {  
 printf("========================\n");  
 printf("= Menu de Opções =\n");  
 printf("========================\n");  
 ...  
} while (opcao != 10);  
```  
A interface do menu é organizada para ser clara e amigável, com opções numeradas e descrições das funcionalidades.**

**4. Entrada do Usuário:**

**```c  
scanf("%d", &opcao);  
```  
O programa solicita que o usuário digite um número correspondente à sua escolha no menu. A entrada é armazenada na variável `opcao`.**

**5. Controle de Fluxo:**

**A estrutura `switch` direciona o fluxo do programa para a função correspondente à escolha do usuário.**

**Exemplo:  
```c  
case 1:  
 gerarVendas();  
 break;  
```  
Caso o usuário escolha a opção `1`, a função `gerarVendas()` será chamada para iniciar o processo de realizar uma venda. O comando `break` evita que o fluxo continue para outros casos.**

**Tabela com as opções do menu:**

|  |  |  |
| --- | --- | --- |
| **Opção** | **Função Relacionada** | **Descrição** |
| **1** | **gerarVendas()** | **Inicia o processo de realizar uma venda.** |
| **2** | **consultarVendas()** | **Permite consultar vendas realizadas anteriormente.** |
| **3** | **exibirProdutos()** | **Exibe todos os produtos cadastrados no sistema.** |
| **4** | **cadastrarProduto()** | **Permite cadastrar um novo produto no sistema.** |
| **5** | **gerarRelatorios()** | **Gera relatórios com informações detalhadas sobre produtos e vendas.** |
| **6** | **alterarProduto()** | **Permite editar as informações de um produto já cadastrado.** |
| **7** | **creditos()** | **Exibe os créditos do programa, como informações sobre os desenvolvedores.** |
| **10** | **-** | **Sai do sistema, encerrando o programa.** |

**6. Condição de Saída:**

**O laço `do-while` verifica continuamente a condição:  
```c  
while (opcao != 10);  
```  
Quando o usuário digita `10`, o programa encerra a execução e retorna `0`, indicando uma finalização bem-sucedida.**

# Documentação da Função cadastrarProduto()

Esta função é responsável por cadastrar um produto em um arquivo de texto que funciona como banco de dados para armazenar informações dos produtos. Ela realiza validações em todas as entradas do usuário, grava os dados no arquivo e oferece a opção de cancelar o cadastro.

## Descrição do Código

### 1. Limpeza da Tela e Inicialização

A função começa limpando a tela do terminal com a função `limparTela()`. Em seguida, exibe uma mensagem de cabeçalho para indicar que o cadastro de produtos será iniciado. O arquivo `produtos.txt` é aberto no modo de adição (`a`), permitindo que novos dados sejam anexados ao arquivo existente.

### 2. Validação do Código do Produto

A entrada do código do produto é feita em um loop que garante que o código seja um número inteiro positivo. Além disso, a função `compararCodigoProduto()` verifica se o código já existe no banco de dados. Caso o código seja inválido ou duplicado, o usuário será solicitado a inserir outro código.

### 3. Entrada de Dados para o Produto

As informações restantes do produto (nome, marca, estoque, custo e preço de venda) são solicitadas ao usuário. Cada entrada passa por validações específicas para evitar dados inválidos, como estoques ou custos negativos. O nome e a marca do produto são lidos com a função `fgets`, e os valores numéricos são verificados com condicionais.

### 4. Cálculo do Lucro

O lucro bruto do produto é calculado como a diferença entre o preço de venda e o custo. O valor é exibido ao usuário para confirmar os dados antes da gravação no arquivo.

### 5. Confirmação e Gravação no Arquivo

O usuário é questionado se deseja finalizar o cadastro. Caso a resposta seja positiva, os dados do produto são gravados no arquivo `produtos.txt` no formato delimitado por ponto e vírgula (;), facilitando a utilização do arquivo como CSV. Se o usuário optar por cancelar, o cadastro é interrompido.

### 6. Encerramento da Função

Após salvar ou cancelar o cadastro, o arquivo é fechado e a tela é limpa. O programa retorna ao menu principal.

## Trecho de Código da Função

void cadastrarProduto()  
{  
 limparTela();  
 printf("========================\n");  
 printf("Cadastro de produtos:\n");  
 printf("========================\n");  
  
 FILE \*arquivo;  
 arquivo = fopen("produtos.txt", "a");  
  
 if (arquivo == NULL)  
 {  
 printf("Erro ao abrir o arquivo de produtos!\n");  
 return;  
 }  
  
 Produto produto;  
  
 do  
 {  
 printf("\nDigite o código do produto: ");  
 if (scanf("%d", &produto.codigo) != 1 || produto.codigo <= 0)  
 {  
 printf("Entrada inválida! O código deve ser um número inteiro positivo.\n");  
 limparBuffer();  
 }  
 else if (compararCodigoProduto(produto.codigo))  
 {  
 printf("Código já existente! Digite outro código!\n");  
 }  
 else  
 {  
 break;  
 }  
 } while (1);  
  
 limparBuffer();  
 printf("\nDigite o nome do produto:");  
 fgets(produto.nome\_produto, 50, stdin);  
 produto.nome\_produto[strcspn(produto.nome\_produto, "\n")] = 0;  
  
 printf("\nDigite a marca do produto:");  
 fgets(produto.marca, 50, stdin);  
 produto.marca[strcspn(produto.marca, "\n")] = 0;  
  
 do  
 {  
 printf("\nDigite o estoque do produto:");  
 if (scanf("%d", &produto.estoque) != 1 || produto.estoque <= 0)  
 {  
 printf("\nEntrada inválida! O estoque não pode ser negativo.\n");  
 limparBuffer();  
 }  
 else  
 {  
 break;  
 }  
 } while (1);  
  
 limparBuffer();  
 do  
 {  
 printf("\nDigite o custo do produto:");  
 if (scanf("%f", &produto.custo) <= 0)  
 {  
 printf("\nEntrada inválida! O custo do produto não pode ser negativo;\n");  
 limparBuffer();  
 }  
 else  
 {  
 break;  
 }  
 } while (1);  
  
 limparBuffer();  
 printf("\nDigite o preço de venda do produto:");  
 scanf("%f", &produto.venda);  
 getchar();  
  
 printf("\nLucro bruto obtido na venda: ");  
 produto.lucro = produto.venda - produto.custo;  
 printf("R$%.2f\n", produto.lucro);  
  
 char resposta;  
 do  
 {  
 printf("\nDeseja finalizar o cadastro? (S para Sim, N para Não): ");  
 resposta = getchar();  
 limparBuffer();  
  
 if (resposta == 'S' || resposta == 's')  
 {  
 fprintf(arquivo, "%d; %s; %s; %d; %.2f; %.2f; %.2f\n",  
 produto.codigo,  
 produto.nome\_produto,  
 produto.marca, produto.estoque,  
 produto.custo,  
 produto.venda,  
 produto.lucro);  
  
 printf("\nProduto %s cadastrado com sucesso!\n", produto.nome\_produto);  
 fclose(arquivo);  
 break;  
 }  
 else if (resposta == 'N' || resposta == 'n')  
 {  
 printf("\nCadastro cancelado.\n");  
 fclose(arquivo);  
 return;  
 }  
 else  
 {  
 printf("\nResposta inválida! Por favor, digite 'S' para Sim ou 'N' para Não.\n");  
 }  
 } while (resposta != 'S' && resposta != 's' && resposta != 'N' && resposta != 'n');  
  
 sairComQualquerTecla();  
 limparTela();  
}

**Função abrirArquivoProdutos**

Essa função recebe o ponteiro do arquivo de banco de dados e é utilizada para abrir o arquivo. Caso o programa não encontre um arquivo correspondente, uma mensagem de erro será exibida.

c

void abrirArquivoProdutos(FILE \*arquivo)

{

arquivo = fopen("produtos.txt", "a");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de produtos!\n");

return;

}

}

**Função limparBuffer**

Função para limpar o buffer, utilizada para evitar entradas incorretas da memória e gravações em variáveis erradas.

c

void limparBuffer()

{

// Consome os caracteres restantes no buffer até encontrar o '\n' ou EOF

while (getchar() != '\n' && getchar() != EOF)

;

}

Função sairComQualquerTecla

Essa função recebe qualquer tecla como input.

void sairComQualquerTecla()

{

printf("Aperte qualquer tecla para sair: \n");

while (1)

{

if (\_kbhit())

{

char tecla = getch();

break;

}

}

}

Função limparTela

Essa função usa comandos de C para interagir com o terminal do sistema operacional e limpar a tela.

c

void limparTela()

{

#ifdef \_WIN32 // Verifica se o sistema é Windows ou não

system("cls");

#else

system("clear");

#endif

}

Função obterInteiroPositivo

Essa função é utilizada no programa para editar produtos, recebendo dados do usuário e atualizando o banco com novos valores. Os dados digitados pelo usuário são armazenados como string e depois convertidos para inteiros.

**int obterInteiroPositivo(const char \*mensagem)**

{

int valor;

char armazenador[50];

// Os dados digitados pelo usuário são armazenados em uma string, e depois transformados em integers

do

{

printf("%s", mensagem);

if (fgets(armazenador, sizeof(armazenador), stdin))

{

if (armazenador[0] == '\n')

break; // Mantém o valor atual caso o usuário digite Enter

if (sscanf(armazenador, "%d", &valor) != 1 || valor <= 0)

{

printf("Entrada inválida! O valor deve ser positivo.\n");

}

else

{

return valor;

}

}

limparBuffer();

} while (1);

return -1; // Retorna valor inválido caso o usuário deseje manter o valor

}

**Função obterFloatPositivo**

Essa função é similar à anterior, mas transforma os dados capturados em variáveis do tipo float. Ela garante que o valor fornecido seja um número positivo.

c

float obterFloatPositivo(const char \*mensagem)

{

float valor;

char armazenador[50];

do

{

printf("%s", mensagem);

if (fgets(armazenador, sizeof(armazenador), stdin))

{

if (armazenador[0] == '\n')

break; // Mantém o valor atual

if (sscanf(armazenador, "%f", &valor) != 1 || valor <= 0)

{

printf("Entrada inválida! O valor deve ser positivo.\n");

}

else

{

return valor;

}

}

limparBuffer();

} while (1);

return -1; // Retorna valor inválido caso o usuário deseje manter o valor

}

**Função alterarProduto**

Essa função permite alterar quaisquer informações de um produto no banco de dados, substituindo as informações na mesma linha, com exceção do código do produto, que não pode ser alterado.

c

void alterarProduto()

{

FILE \*ptr\_arquivo\_produtos;

char linha[MAX];

char cod\_produto[MAX];

char result[MAX];

int i = 0;

char reescrita[MAX];

Produto produto;

exibirProdutos(); // Exibe a lista de produtos cadastrados

printf("\nInforme o codigo do produto que deseja alterar:\n");

scanf("%s", cod\_produto);

limparBuffer();

// Abre o arquivo de produtos. Caso não exista um arquivo, o programa retorna uma mensagem de erro.

ptr\_arquivo\_produtos = fopen("produtos.txt", "r+");

if (ptr\_arquivo\_produtos == NULL)

{

printf("Erro na abertura do arquivo.\n");

return;

}

while (fgets(linha, MAX, ptr\_arquivo\_produtos)) // Armazena o input do banco em uma string

{

i = 0;

while (linha[i] != ';' && linha[i] != '\0') // Continua lendo os caracteres até encontrar ';'

{

result[i] = linha[i];

i++;

}

result[i] = '\0'; // Inclui o fim da string

if (strcmp(cod\_produto, result) == 0) // Compara o código do produto digitado com a string da memória

{

printf("========================\n");

printf("Produto encontrado!\n");

printf("========================\n");

printf("## ALTERANDO PRODUTO ##\n");

printf("========================\n");

printf("Informe apenas os campos que deseja alterar\ncaso não queira, deixe em branco!\n");

// Nome do produto

printf("\nDigite o nome do produto (pressione Enter para manter o valor atual): ");

fgets(produto.nome\_produto, 50, stdin);

produto.nome\_produto[strcspn(produto.nome\_produto, "\n")] = 0;

sprintf(reescrita, "%s; %s", result, produto.nome\_produto);

// Marca do produto

printf("\nDigite a marca do produto (pressione Enter para manter o valor atual): ");

fgets(produto.marca, 50, stdin);

produto.marca[strcspn(produto.marca, "\n")] = 0;

sprintf(reescrita, "%s; %s", reescrita, produto.marca);

// Estoque

produto.estoque = obterInteiroPositivo("\nDigite o estoque do produto (pressione Enter para manter o valor atual): ");

if (produto.estoque != -1)

{

sprintf(reescrita, "%s; %d", reescrita, produto.estoque);

}

// Custo

produto.custo = obterFloatPositivo("\nDigite o custo do produto (pressione Enter para manter o valor atual): ");

if (produto.custo != -1)

{

sprintf(reescrita, "%s; %.2f", reescrita, produto.custo);

}

// Preço de venda

produto.venda = obterFloatPositivo("\nDigite o preço de venda do produto (pressione Enter para manter o valor atual): ");

if (produto.venda != -1)

{

sprintf(reescrita, "%s; %.2f", reescrita, produto.venda);

}

// Lucro

produto.lucro = produto.venda - produto.custo;

sprintf(reescrita, "%s; %.2f", reescrita, produto.lucro);

printf("Lucro bruto obtido na venda: R$%.2f\n", produto.lucro);

fseek(ptr\_arquivo\_produtos, -strlen(linha), SEEK\_CUR);

fprintf(ptr\_arquivo\_produtos, "%s", reescrita);

// Se a nova linha for menor que a original, preenche o restante com espaços.

// Isso é necessário porque o programa precisa sobrescrever a linha de dados anterior com as novas informações,

// e caso a anterior seja maior, os caracteres extras não serão deletados

int tamanho\_novo = strlen(reescrita);

int tamanho\_antigo = strlen(linha);

if (tamanho\_novo < tamanho\_antigo)

{

int j;

for (j = 0; j < (tamanho\_antigo - tamanho\_novo); j++)

{

fputc(' ', ptr\_arquivo\_produtos); // Inclui espaços para apagar os caracteres restantes na linha antiga

}

}

break;

}

}

fclose(ptr\_arquivo\_produtos); // Fecha o arquivo

}

Função obterQuantidadeProduto

Esta função é utilizada ao realizar uma venda para garantir que a quantidade de produto solicitada não exceda o estoque disponível. Ela solicita ao usuário que informe a quantidade de produto e valida a entrada.

int obterQuantidadeProduto(int estoque)

c

{

int qtd\_produto;

do

{

printf("Digite a quantidade que será vendida: ");

if (scanf("%d", &qtd\_produto) != 1 || qtd\_produto <= 0 || qtd\_produto > estoque)

{

printf("\nQuantidade inválida ou superior ao estoque disponível!\n");

while (getchar() != '\n')

; // Limpa o buffer

}

else

{

break;

}

} while (1);

return qtd\_produto;

}

**Função registrarVenda**

Essa função recebe os valores armazenados em algumas variáveis durante o processo de venda e os armazena em outras variáveis para posterior gravação da venda.

c

void registrarVenda(Vendas \*venda, int codigo, const char \*nome\_produto, int qtd\_produto, float venda\_valor)

{

venda->codigo\_produto = codigo;

strcpy(venda->nome\_produto, nome\_produto);

venda->qtd\_produto = qtd\_produto;

venda->venda = venda\_valor;

venda->valor\_total = qtd\_produto \* venda\_valor;

}

**Função gerarVendas**

Esta função exibe o menu de vendas e permite que o usuário escolha entre consultar produtos, gerar vendas ou voltar ao menu anterior. Ela também manipula a abertura do arquivo de produtos para garantir que o sistema tenha acesso às informações dos produtos.

c

void gerarVendas()

{

limparTela();

FILE \*arquivo = fopen("produtos.txt", "r"); // Abre o arquivo de produtos

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de produtos!\n");

return;

}

int op;

do

{

printf("\n===== PDV =====\n"); // Menu de vendas

printf(

"[1] - Consultar produtos\n"

"[2] - Gerar vendas\n"

"[3] - Voltar\n");

printf("\nEscolha: ");

scanf("%d", &op);

switch (op)

{

case 1:

exibirProdutos();

break;

case 2:

venderProdutos();

break;

case 3:

break;

}

} while (op != 3);

fclose(arquivo);

limparTela();

}

**Função venderProdutos**

A função venderProdutos permite que o usuário realize vendas de múltiplos produtos, verificando os dados do estoque e gravando as informações da venda, como a data, hora e a forma de pagamento.

c

void venderProdutos()

{

limparTela();

int n;

const char \*nome\_arquivo = "numero\_nota.txt";

int numero\_nota = ler\_numero\_nota(nome\_arquivo);

FILE \*arquivo = fopen("produtos.txt", "r"); // Abre o arquivo de produtos

FILE \*vendas = fopen("vendas.txt", "a+");

if (arquivo == NULL || vendas == NULL)

{

printf("Erro ao abrir o arquivo de produtos!\n");

return;

}

// Função da biblioteca <time.h> para obter data e hora atuais

time\_t t = time(NULL);

struct tm tm = \*localtime(&t);

char data[20], hora[20];

strftime(data, sizeof(data), "%d/%m/%Y", &tm);

strftime(hora, sizeof(hora), "%H:%M:%S", &tm);

// Obter a quantidade de produtos a serem vendidos

printf("Informe quantos produtos diferentes deseja vender: ");

scanf("%d", &n);

int cod\_produtos[n];

int i;

for (i = 0; i < n; i++)

{

do

{

printf("Informe o codigo do %dº produto que irá realizar a venda: ", i + 1);

if (scanf("%d", &cod\_produtos[i]) != 1 || !compararCodigoProduto(cod\_produtos[i]))

{

printf("Código do produto inválido ou produto não encontrado!\n");

while (getchar() != '\n')

; // Limpa o buffer

}

else

{

break;

}

} while (1);

}

// Variáveis para processamento de venda

int codigo, estoque, formaPagamento;

float custo, venda, lucro;

char nome\_produto[50], marca[50];

Vendas vetor\_vendas[n];

int contador = 0;

// Processar cada produto

while (fscanf(arquivo, "%d; %49[^;]; %49[^;]; %d; %f; %f; %f\n",

&codigo, nome\_produto, marca, &estoque, &venda, &custo, &lucro) == 7)

{

int i;

for (i = 0; i < n; i++)

{

if (cod\_produtos[i] == codigo)

{

int qtd\_produto = obterQuantidadeProduto(estoque);

if (qtd\_produto > 0)

{

registrarVenda(&vetor\_vendas[contador], codigo, nome\_produto, qtd\_produto, venda);

contador++;

}

break;

}

}

}

// Obter a forma de pagamento

limparTela();

printf("\nDigite a forma de pagamento:\n[1] - Dinheiro\n[2] - Cartão débito\n[3] - Cartão crédito: ");

scanf("%d", &formaPagamento);

// Gravar vendas no arquivo

for (i = 0; i < contador; i++)

{

fprintf(vendas, "venda;%d;%s;%d;%.2f;%.2f;%s;%s;%d\n",

numero\_nota, vetor\_vendas[i].nome\_produto,

vetor\_vendas[i].qtd\_produto, vetor\_vendas[i].venda,

vetor\_vendas[i].valor\_total, data, hora, formaPagamento);

}

// Exibir mensagem de sucesso e a nota fiscal

printf("\nVenda realizada com sucesso!\n");

exibirNotaFiscal(vetor\_vendas, contador, formaPagamento);

fclose(arquivo);

fclose(vendas);

}

**Função: exibirCupomHeader**

**O que ela faz:**  
Essa função exibe o cabeçalho do cupom fiscal, incluindo o nome da empresa, CNPJ, endereço, e informações do cupom, como data, hora, número da nota fiscal e operador. Ela organiza as informações de forma formatada para exibição no arquivo de saída.

c

// função para exibir o cabeçalho do cupom fiscal

void exibirCupomHeader(FILE \*arquivo, const char \*data, const char \*hora, int numero\_nota)

{

fprintf(arquivo, "===============================================\n");

fprintf(arquivo, " CUPOM FISCAL ELETRÔNICO \n");

fprintf(arquivo, "===============================================\n");

fprintf(arquivo, "Empresa: LUIZ INÁCIO LTDA\n");

fprintf(arquivo, "CNPJ: 66.666.666/0001-66\n");

fprintf(arquivo, "Endereço: Congresso Nacional, 666 - Esquerda\n");

fprintf(arquivo, "IE:000.000000.0000 CEP: 66666-666\n");

fprintf(arquivo, "-----------------------------------------------\n");

fprintf(arquivo, "Data: %s Hora: %s Cupom: %d\n", data, hora, numero\_nota);

fprintf(arquivo, "Operador: Alexandre de Morais\n");

fprintf(arquivo, "===============================================\n");

fprintf(arquivo, "CÓDIGO DESCRIÇÃO QTD UNIT. TOTAL \n");

fprintf(arquivo, "-----------------------------------------------\n");

}

**Função: exibirDetalhesProduto**

**O que ela faz:**  
Essa função exibe os detalhes de cada produto vendido, como o código do produto, nome, quantidade, preço unitário e o valor total da venda do produto.

c

void exibirDetalhesProduto(FILE \*arquivo, Vendas \*venda)

{

fprintf(arquivo, "%-7d %-10s %-2d %-8.2f %-10.2f\n",

venda->codigo\_produto, venda->nome\_produto, venda->qtd\_produto, venda->venda, venda->valor\_total);

}

Função: exibirCupomFooter

**O que ela faz:**  
Essa função exibe o rodapé do cupom fiscal, incluindo o total de itens, o valor total da venda, o valor do imposto pago (considerando 30% sobre o valor total) e a forma de pagamento escolhida (dinheiro, débito ou crédito). Ela também exibe a chave de acesso gerada e uma mensagem de agradecimento.

c

// função para exibir o rodapé do cupom fiscal

void exibirCupomFooter(FILE \*arquivo, int totalItens, float valorTotal, int tipo, const char \*digitos)

{

fprintf(arquivo, "-----------------------------------------------\n");

fprintf(arquivo, "QTD. total de itens %d\n", totalItens);

fprintf(arquivo, "Valor total do desconto R$ 0,00\n");

fprintf(arquivo, "TOTAL R$ %.2f\n", valorTotal);

fprintf(arquivo, "-----------------------------------------------\n");

fprintf(arquivo, "IMPOSTO TOTAL PAGO: R$ %.2f\n", valorTotal \* 0.30);

fprintf(arquivo, "-----------------------------------------------\n");

// Forma de pagamento, dinheiro, cartao de débito ou crédito

if (tipo == 1)

{

fprintf(arquivo, "FORMA DE PAGAMENTO: DINHEIRO \n");

}

else if (tipo == 2)

{

fprintf(arquivo, "FORMA DE PAGAMENTO: CARTÃO DÉBITO \n");

}

else if (tipo == 3)

{

fprintf(arquivo, "FORMA DE PAGAMENTO: CARTÃO CRÉDITO \n");

}

fprintf(arquivo, "-----------------------------------------------\n");

fprintf(arquivo, " Chave de acesso\n");

fprintf(arquivo, " %s\n", digitos); // Exibe os 44 dígitos gerados

fprintf(arquivo, "===============================================\n");

fprintf(arquivo, " Obrigado por comprar conosco! \n");

fprintf(arquivo, "===============================================\n");

}

**Função: exibirNotaFiscal**

**O que ela faz:**  
Essa função exibe e imprime a nota fiscal completa, incluindo informações sobre a venda (como o número da nota, data, hora, itens vendidos, valores, impostos e forma de pagamento), gera uma chave de acesso e a imprime tanto no console quanto em um arquivo de texto.

Além disso, ela organiza as informações em um formato legível e permite a impressão do cupom fiscal em um arquivo específico. O código também inclui um arquivo gerado com os detalhes da venda e o processo de incremento do número da nota fiscal.

c

// função para exibir e imprimir a nota fiscal inteira, incluindo algumas informações geradas ao longo da venda do produto

void exibirNotaFiscal(Vendas \*ptr\_vetor\_vendas, int tamanho, int tipo)

{

limparTela();

const char \*nome\_arquivo = "numero\_nota.txt";

if (incrementar\_nota(nome\_arquivo) != 0)

{

fprintf(stderr, "Falha ao incrementar o número da nota.\n");

return;

}

int numero\_nota = ler\_numero\_nota(nome\_arquivo);

if (numero\_nota == -1)

{

fprintf(stderr, "Falha ao ler o número da nota.\n");

return;

}

char digitos[45];

gerarChaveAcesso(digitos);

// Obter data e hora atuais

time\_t t = time(NULL);

struct tm tm = \*localtime(&t);

char data[20], hora[20];

strftime(data, sizeof(data), "%d/%m/%Y", &tm);

strftime(hora, sizeof(hora), "%H:%M:%S", &tm);

// Exibir cabeçalho do cupom fiscal

printf("===============================================\n");

printf(" CUPOM FISCAL ELETRÔNICO \n");

printf("===============================================\n");

printf("Empresa: LUIZ INÁCIO LTDA\n");

printf("CNPJ: 66.666.666/0001-66\n");

printf("Endereço: Congresso Nacional, 666 - Esquerda\n");

printf("IE:000.000000.0000 CEP: 66666-666\n");

printf("-----------------------------------------------\n");

printf("Data: %s Hora: %s Cupom: %d\n", data, hora, numero\_nota);

printf("Operador: Alexandre de Morais\n");

printf("===============================================\n");

printf("CÓDIGO DESCRIÇÃO QTD UNIT. TOTAL \n");

printf("-----------------------------------------------\n");

// Processar e exibir cada item vendido

int totalItens = 0;

float valorTotal = 0;

int i;

for (i = 0; i < tamanho; i++)

{

Vendas venda = ptr\_vetor\_vendas[i];

totalItens += venda.qtd\_produto;

valorTotal += venda.valor\_total;

printf("%-7d %-10s %-2d %-8.2f %-10.2f\n",

venda.codigo\_produto, venda.nome\_produto, venda.qtd\_produto, venda.venda, venda.valor\_total);

}

printf("-----------------------------------------------\n");

printf("QTD. total de itens %d\n", totalItens);

printf("Valor total do desconto R$ 0,00\n");

printf("TOTAL R$ %.2f\n", valorTotal);

printf("-----------------------------------------------\n");

printf("IMPOSTO TOTAL PAGO: R$ %.2f\n", valorTotal \* 0.30);

printf("-----------------------------------------------\n");

// Exibir forma de pagamento

if (tipo == 1)

{

printf("FORMA DE PAGAMENTO: DINHEIRO \n");

}

else if (tipo == 2)

{

printf("FORMA DE PAGAMENTO: CARTÃO DÉBITO \n");

}

else if (tipo == 3)

{

printf("FORMA DE PAGAMENTO: CARTÃO CRÉDITO \n");

}

// Exibir chave de acesso

printf("Chave de acesso: %s\n", digitos);

printf("===============================================\n");

printf(" Obrigado por comprar conosco! \n");

printf("===============================================\n");

// Salvar o cupom fiscal em um arquivo

char nomeArquivo[100];

snprintf(nomeArquivo, sizeof(nomeArquivo), "C:\\Área de Trabalho\\Curso C\\projeto final adam\\NOTAS\\nota\_fiscal\_%d.txt", time(NULL));

FILE \*arquivo = fopen(nomeArquivo, "w");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo!\n");

return;

}

// Gravar os dados no arquivo

exibirCupomHeader(arquivo, data, hora, numero\_nota);

for (i = 0; i < tamanho; i++)

{

exibirDetalhesProduto(arquivo, &ptr\_vetor\_vendas[i]);

}

exibirCupomFooter(arquivo, totalItens, valorTotal, tipo, digitos);

fclose(arquivo);

printf("Cupom fiscal gerado em '%s'\n", nomeArquivo);

printf("Aperte qualquer tecla para sair: \n");

// Aguarda a tecla pressionada

while (!\_kbhit())

{

}

getch();

limparTela();

}

**Função gerarChaveAcesso**

Essa função gera um número aleatório de 44 dígitos que pode ser usado como uma chave de acesso para a nota fiscal.

**Explicação do código:**

* A função percorre um laço de 44 iterações, onde em cada uma delas um número aleatório de 0 a 9 é gerado e adicionado à variável sequencia.
* O número é gerado pela expressão rand() % 10, que garante que o valor esteja entre 0 e 9.
* Ao final, o caractere de terminação \0 é inserido no final da string para indicar o fim.

c

// função para gerar um número aleatorio para a nota fiscal

void gerarChaveAcesso(char \*sequencia)

{

int i;

for (i = 0; i < 44; i++)

{

sequencia[i] = '0' + (rand() % 10); // gerador de números aleatórios

}

sequencia[44] = '\0'; // finaliza a string

}

**Função exibirProdutos**

Essa função lê os dados dos produtos cadastrados de um arquivo (no formato CSV) e exibe as informações dos produtos de maneira formatada no console. O arquivo de entrada contém os dados dos produtos separados por ponto e vírgula (;).

**Explicação do código:**

* O arquivo produtos.txt é aberto para leitura.
* A função lê linha por linha do arquivo e usa a função fscanf para capturar os dados do produto (código, nome, marca, estoque, custo, preço de venda e lucro).
* O formato de leitura considera que os dados estão separados por ponto e vírgula (;), e usa a especificação de formato %d para inteiros, %f para floats e %49[^;] para strings de até 49 caracteres.
* A função imprime as informações de cada produto em uma tabela organizada.
* Se o arquivo não puder ser aberto, uma mensagem de erro é exibida.

c

// função que escaneia o arquivo de produtos linha por linha, armazenando os valores entre ; nas variaveis da struct produtos e exibindo para o usuário

void exibirProdutos()

{

limparTela();

FILE \*arquivo;

arquivo = fopen("produtos.txt", "r");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de produtos!\n");

}

printf("========================\n");

printf("=== PRODUTOS CADASTRADOS ===\n");

printf("========================\n");

int codigo, estoque;

float custo, venda, lucro;

char nome\_produto[50], marca[50];

printf("----------------------------------------------------------------------------------");

printf("\n%-7s|%-15s|%-15s|%-8s|%-10s|%-10s|%-10s\n", "Código", "Nome", "Marca", "Estoque", "Custo", " Venda", " Lucro");

printf("----------------------------------------------------------------------------------\n");

while (fscanf(arquivo, "%d; %49[^;]; %49[^;]; %d; %f; %f; %f\n", &codigo, nome\_produto, marca, &estoque, &custo, &venda, &lucro) == 7)

{

printf("%-7d|%-15s|%-15s|%-8d|R$%-9.2f|R$%-9.2f|R$%-9.2f\n", codigo, nome\_produto, marca, estoque, custo, venda, lucro);

printf("----------------------------------------------------------------------------------\n");

}

fclose(arquivo);

}

**Função: compararCodigoProduto**

**O que ela faz:**  
Essa função compara o código do produto informado pelo usuário com os códigos dos produtos armazenados no arquivo produtos.txt. Ela escaneia o arquivo produto por produto, e se encontrar um código que corresponda ao código informado, retorna 1 (indicando que o código foi encontrado). Caso contrário, retorna 0 (indicando que o código não foi encontrado).

c

// função para escanear o arquivo de produtos e comparar o código do produto digitado pelo usuário até encontrar um match

**int compararCodigoProduto(int numero)**

{

FILE \*arquivo;

arquivo = fopen("produtos.txt", "a");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de produtos!\n");

}

// abrirArquivoProdutos(arquivo);

int codigo, estoque;

float custo, venda, lucro;

char nome\_produto[50], marca[50];

while (fscanf(arquivo, "%d; %49[^;]; %49[^;]; %d; %f; %f; %f\n", &codigo, nome\_produto, marca, &estoque, &custo, &venda, &lucro) == 7) // escaneia cada linha do produtos.txt armazenando os valores entre ; nas variaveis

{

if (numero == codigo)

{

fclose(arquivo);

return 1;

}

}

fclose(arquivo);

return 0;

}

**abrirArquivoCsv**

**O que ela faz:**  
Essa função tenta abrir um arquivo CSV em modo de escrita ("w"). Caso o arquivo não seja aberto corretamente, exibe uma mensagem de erro. No entanto, o parâmetro csv não está sendo utilizado corretamente, pois a função recebe o ponteiro por valor e não altera o ponteiro da função chamadora.

c

// função para abrir o arquivo em formato csv

void abrirArquivoCsv(FILE \*csv, char path[])

{

csv = fopen(path, "w");

if (csv == NULL)

{

printf("Erro ao abrir o arquivo CSV de produtos!\n");

return;

}

}

**gerarRelatorioProdutos**

**O que ela faz:**  
Essa função gera um relatório de produtos em formato CSV. Ela lê os dados do arquivo produtos.txt, extrai as informações sobre os produtos, e escreve no arquivo CSV localizado em um local definido. O cabeçalho do CSV é configurado com os campos Código, Nome, Marca, Estoque, Custo, Venda, e Lucro, e depois os dados dos produtos são gravados.

c

// função para salvar o arquivo de produtos em formato .csv

void gerarRelatorioProdutos()

{

limparTela();

FILE \*arquivo;

arquivo = fopen("produtos.txt", "a");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de produtos!\n");

return;

}

// abrirArquivoProdutos(arquivo);

FILE \*csv;

csv = fopen("C:\\Área de Trabalho\\Curso C\\projeto final adam\\RELATORIOS\\produtos.csv", "w");

if (csv == NULL)

{

printf("Erro ao abrir o arquivo CSV de produtos!\n");

return;

}

// abrirArquivoCsv(csv, "C:\\Área de Trabalho\\Curso C\\projeto final adam\\RELATORIOS\\produtos.csv");

fprintf(csv, "Código;Nome;Marca;Estoque;Custo;Venda;Lucro\n");

int codigo, estoque;

float custo, venda, lucro;

char nome\_produto[50], marca[50];

while (fscanf(arquivo, "%d; %49[^;]; %49[^;]; %d; %f; %f; %f\n", &codigo, nome\_produto, marca, &estoque, &custo, &venda, &lucro) == 7)

{

fprintf(csv, "%d;%s;%s;%d;%.2f;%.2f;%.2f\n", codigo, nome\_produto, marca, estoque, custo, venda, lucro);

}

fclose(arquivo);

fclose(csv);

printf("\nRelatório gerado com sucesso em 'C:\\Área de Trabalho\\Curso C\\projeto final adam\\RELATORIOS\\produtos.csv'!\n");

}

**abrirArquivoVendas**

**O que ela faz:**  
Essa função tenta abrir o arquivo vendas.txt em modo de leitura ("r"). Se não conseguir abrir o arquivo, ela exibe uma mensagem de erro. No entanto, a função tem um pequeno problema: o ponteiro arquivo é passado por valor, então a alteração nele não afeta o ponteiro na função chamadora.

c

// função para abrir o arquivo vendas.txt

void abrirArquivoVendas(FILE \*arquivo)

{

arquivo = fopen("vendas.txt", "r");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de vendas!\n");

return;

}

}

**gerarRelatorioVenda**

**O que ela faz:**  
Essa função gera um relatório das vendas, que é exportado para um arquivo CSV. Ela faz isso da seguinte maneira:

* Abre o arquivo vendas.txt em modo leitura.
* Cria e abre um arquivo CSV na localização especificada.
* Escreve o cabeçalho no arquivo CSV.
* Lê os dados do arquivo vendas.txt linha por linha e os escreve no arquivo CSV.
* Após processar os dados, ela fecha ambos os arquivos (vendas.txt e o arquivo CSV).
* Exibe uma mensagem de sucesso, indicando que o relatório foi gerado corretamente.

c

// função para gerar o relatório de vendas

void gerarRelatorioVenda()

{

limparTela();

FILE \*arquivo;

arquivo = fopen("vendas.txt", "r");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de vendas!\n");

return;

}

// abrirArquivoVendas(arquivo);

FILE \*csv;

csv = fopen("C:\\Área de Trabalho\\Curso C\\projeto final adam\\RELATORIOS\\relatorios\_vendas.csv", "w");

if (csv == NULL)

{

printf("Erro ao abrir o arquivo CSV de produtos!\n");

return;

}

// abrirArquivoCsv(csv, "C:\\Área de Trabalho\\Curso C\\projeto final adam\\RELATORIOS\\relatorios\_vendas.csv");

fprintf(csv, "Numero da Venda; Nome do Produto; Quantidade; Custo Unitario; Valor de Venda; Data; Hora; Tipo de Pagamento\n");

int numero\_venda, quantidade, tipo\_pagamento;

float custo, venda;

char nome\_produto[50], data[20], hora[20];

while (fscanf(arquivo, "venda;%d;%[^;];%d;%f;%f;%[^;];%[^;];%d\n",

&numero\_venda, nome\_produto, &quantidade, &custo, &venda, data, hora, &tipo\_pagamento) == 8)

{

fprintf(csv, "%d; %s; %d; %.2f; %.2f; %s; %s; %d\n",

numero\_venda, nome\_produto, quantidade, custo, venda, data, hora, tipo\_pagamento);

}

fclose(arquivo);

fclose(csv);

printf("Relatório gerado com sucesso em 'C:\\Área de Trabalho\\Curso C\\projeto final adam\\RELATORIOS\\relatorios\_vendas.csv'!\n");

}

**Função gerarRelatorios**

A função gerarRelatorios serve para permitir que o usuário escolha qual relatório deseja gerar: de produtos ou de vendas. Ela apresenta um menu para o usuário, recebe a escolha e, com base nessa escolha, chama a função correspondente para gerar o relatório.

**Fluxo de execução:**

1. **Exibição do menu:**  
   A função começa limpando a tela (com a função limparTela()), em seguida, exibe um menu com três opções:
   * Relatório de produtos
   * Relatório de vendas
   * Voltar
2. **Entrada de dados:**  
   O usuário insere a opção desejada.
3. **Processamento da escolha:**
   * Caso o usuário escolha a opção 1, a função gerarRelatorioProdutos() é chamada para gerar o relatório de produtos.
   * Caso o usuário escolha a opção 2, a função gerarRelatorioVenda() é chamada para gerar o relatório de vendas.
   * Caso o usuário escolha a opção 3, o menu retorna sem fazer nada.
   * Se a opção for inválida (qualquer valor diferente de 1, 2 ou 3), uma mensagem de erro é exibida.

c

void gerarRelatorios()

{

limparTela();

int opcao;

printf("========================\n");

printf("\nGerar relatórios\n");

printf("========================\n");

printf("\nEscolha o relatório\n");

printf("[1] - Relatório de produtos\n"

"[2] - Relatório de vendas\n"

"[3] - Voltar\n");

printf("\nDigite a escolha: ");

scanf("%d", &opcao);

switch (opcao)

{

case 1:

gerarRelatorioProdutos();

break;

case 2:

gerarRelatorioVenda();

break;

case 3:

break;

default:

printf("\nOpção inválida!\n");

}

}

**Função ler\_numero\_nota**

A função ler\_numero\_nota tem o objetivo de ler o número de uma nota fiscal (ou algum tipo de identificador) a partir de um arquivo.

**Fluxo de execução:**

1. **Abertura do arquivo:**  
   A função tenta abrir o arquivo especificado pelo parâmetro nome\_arquivo em modo leitura ("r"). Se falhar, ela imprime uma mensagem de erro e retorna -1.
2. **Leitura do número da nota:**  
   Se o arquivo for aberto corretamente, a função lê o número da nota fiscal usando fscanf e armazena o valor na variável numero\_nota. Se ocorrer um erro na leitura (ou se o formato do arquivo for inesperado), a função imprime uma mensagem de erro, fecha o arquivo e retorna -1.
3. **Fechamento do arquivo:**  
   Após a leitura bem-sucedida, o arquivo é fechado e o número da nota é retornado.

c

int ler\_numero\_nota(const char \*nome\_arquivo)

{

FILE \*ptr\_nota = fopen(nome\_arquivo, "r");

if (ptr\_nota == NULL)

{

fprintf(stderr, "Erro ao abrir o arquivo '%s' para leitura!\n", nome\_arquivo);

return -1;

}

int numero\_nota;

if (fscanf(ptr\_nota, "%d", &numero\_nota) != 1)

{

fprintf(stderr, "Erro ao ler o número da nota do arquivo!\n");

fclose(ptr\_nota);

return -1;

}

fclose(ptr\_nota);

return numero\_nota;

}

**Função incrementar\_nota**

A função incrementar\_nota tem o objetivo de ler o número da nota fiscal de um arquivo, incrementar esse número e salvar o novo número no mesmo arquivo. A função começa com a leitura do número da nota através da função ler\_numero\_nota. Se a leitura falhar, ela retorna -1. Se o número da nota for lido corretamente, a função abre o arquivo em modo de escrita ("w"), escreve o número incrementado e fecha o arquivo.

**Fluxo de execução:**

1. **Leitura do número da nota:**  
   A função chama ler\_numero\_nota para obter o número da nota atual do arquivo. Se houver um erro, ela retorna -1.
2. **Abertura do arquivo:**  
   O arquivo é aberto em modo de escrita ("w"), o que sobrescreve seu conteúdo. Se o arquivo não puder ser aberto, a função exibe uma mensagem de erro e retorna -1.
3. **Incremento e gravação do número:**  
   O número da nota é incrementado em 1, e o novo valor é gravado no arquivo.
4. **Fechamento do arquivo:**  
   O arquivo é fechado e a função retorna 0, indicando sucesso.

**Código:**

c

int incrementar\_nota(const char \*nome\_arquivo)

{

int numero\_nota = ler\_numero\_nota(nome\_arquivo);

if (numero\_nota == -1)

{

return -1;

}

FILE \*ptr\_nota = fopen(nome\_arquivo, "w");

if (ptr\_nota == NULL)

{

fprintf(stderr, "Erro ao abrir o arquivo '%s' para escrita!\n", nome\_arquivo);

return -1;

}

fprintf(ptr\_nota, "%d", numero\_nota + 1);

fclose(ptr\_nota);

return 0;

}

**Função consultarVendas**

A função consultarVendas tem o objetivo de exibir as informações de vendas armazenadas em um arquivo de texto (vendas.txt). Ela lê as linhas do arquivo, processa os dados e imprime uma tabela com os detalhes das vendas. Além disso, calcula o total das vendas e exibe esse valor no final.

**Fluxo de execução:**

1. **Abertura do arquivo de vendas:**  
   A função tenta abrir o arquivo vendas.txt em modo leitura ("r"). Se falhar, imprime uma mensagem de erro e retorna.
2. **Leitura das linhas do arquivo:**  
   A função lê cada linha do arquivo usando fgets. Se a linha for vazia ou inválida (menos de 5 caracteres), ela é ignorada. A linha é então processada para extrair os dados.
3. **Processamento da linha:**  
   A função utiliza sscanf para extrair os campos da linha (número da venda, nome do produto, quantidade, custo, valor de venda, data, hora e tipo de pagamento). Se todos os campos forem lidos corretamente (8 campos), os dados são exibidos na tabela. Caso contrário, uma mensagem de erro é exibida.
4. **Acumulação do total das vendas:**  
   A função acumula o valor de venda em total\_vendas para ser exibido ao final.
5. **Exibição do relatório:**  
   A função exibe os detalhes de cada venda em uma tabela e, ao final, exibe o total acumulado de vendas.
6. **Fechamento do arquivo e retorno:**  
   Após a leitura do arquivo, ele é fechado e o usuário é informado sobre o total das vendas.

**Código:**

c

void consultarVendas()

{

limparTela();

FILE \*arquivo;

arquivo = fopen("vendas.txt", "r");

if (arquivo == NULL)

{

printf("Erro ao abrir o arquivo de vendas!\n");

return;

}

printf("========================\n");

printf("Consulta de Vendas:\n");

printf("========================\n");

printf("----------------------------------------------------------------------------------------------------\n");

printf("Numero da Venda | Nome do Produto | Qtde | Custo | Venda | Data | Hora | Tipo Pagamento\n");

printf("----------------------------------------------------------------------------------------------------\n");

int numero\_venda, quantidade, tipo\_pagamento;

float custo, venda, total\_vendas = 0.0;

char linha[200];

char nome\_produto[50], data[20], hora[20];

while (fgets(linha, sizeof(linha), arquivo) != NULL)

{

if (strlen(linha) < 5)

continue;

linha[strcspn(linha, "\n")] = 0; // Remove o caractere de nova linha da string

int campos\_lidos = sscanf(linha, "venda;%d;%49[^;];%d;%f;%f;%19[^;];%19[^;];%d",

&numero\_venda, nome\_produto, &quantidade, &custo, &venda, data, hora, &tipo\_pagamento);

if (campos\_lidos == 8)

{

printf("%15d | %-19s | %4d | %8.2f | %8.2f | %9s | %8s | %14d\n",

numero\_venda, nome\_produto, quantidade, custo, venda, data, hora, tipo\_pagamento);

total\_vendas += venda;

}

else

{

printf("Erro ao processar a linha: %s\n", linha);

}

}

fclose(arquivo);

printf("----------------------------------------------------------------------------------------------------\n");

printf("Total das Vendas: R$%.2f\n", total\_vendas);

printf("Aperte qualquer tecla para sair: \n");

sairComQualquerTecla();

limparTela();

}