```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctvpe.h>
#include "goods.h"
typedef struct Produto // Estrutura para armazenar os dados de um produto
   unsigned long long codigo;
   char nome[26];
   float preco;
   int quantidade;
   int Vendas;
   float Subtotal;
   struct Produto* proximo;
} Produto;
typedef struct
   Produto* produto;
   int quantidade;
   float subtotal;
} ItemCupomFiscal:
void exibirEstoque(Produto* estoque) // Função para exibir o estoque de produtos
   if (estoque == NULL) // Verifica se o estoque está vazio
      printf("\nEstoque vazio\n");
   else
      printf("| Item (Codigo)\t\t| Nome do item\t\t| Valor (Unidade)\t| Estoque\n");
      Produto* atual = estoque; // Ponteiro para percorrer a lista de produtos
      while (atual != NULL) // Enquanto houver produtos na lista
          printf("| \$-101lu\t\t| \$-20s\t| R\$ \$.2f\t\t| \$d\n", atual->codigo, atual->nome, atual->preco, atual->quantidade);
          atual = atual->proximo; // Move para o próximo produto
   }
void cadastrarProduto(Produto** estoque) // Função para cadastrar um novo produto
   int quantidadeProdutos;
   printf("Quantos produtos deseja cadastrar? ");
   scanf("%d", &quantidadeProdutos);
   for (int i = 0; i < quantidadeProdutos; i++)
      Produto* novoProduto = (Produto*)malloc(sizeof(Produto)); // Aloca memória para o novo produto
      novoProduto->proximo = NULL; // Define o próximo como NULL, pois ainda não sabemos onde ele será inserido
      printf("\nDados do produto %d:\n", i+1);
      printf("Código do Item (64 bits): ");
      scanf("%llu", &novoProduto->codigo);
      Produto* atual = *estoque; // Define o produto atual como o início do estoque
      while (atual != NULL)
          if (atual->codigo == novoProduto->codigo) // Verifica se o código do novo produto já existe no estoque
             printf("\nErro!! O código do item já existe. Digite novamente...\n");
             free(novoProduto); // Libera a memória alocada para o novo produto
             novoProduto = NULL; // Define o novo produto como NULL para indicar erro
             break;
          atual = atual->proximo; // Avança para o próximo produto no estoque
      if (novoProduto == NULL) // Se o novo produto for NULL, houve um erro, então volta para a iteração anterior
          continue;
      printf("Nome do Item (até 25 caracteres): ");
      scanf(" %s", novoProduto->nome);
      atual = *estoque;
      while (atual != NULL)
          if (strcmp(atual->nome, novoProduto->nome) == 0) // Verifica se o nome do novo produto já existe no estoque
             printf("\nErro!! O nome do item já existe. Digite novamente...\n");
```

```
free (novoProduto); // Libera a memória alocada para o novo produto
               novoProduto = NULL; // Define o novo produto como NULL para indicar erro
               break;
           atual = atual->proximo; // Avança para o próximo produto no estoque
        if (novoProduto == NULL) // Se o novo produto for NULL, houve um erro, então volta para a iteração anterior
           i --:
           continue;
        if (strlen(novoProduto->nome) == 0 || strlen(novoProduto->nome) > 25) // Verifica se o nome possui tamanho adequado
           printf("\nErro!! O nome do item deve ter entre 1 e 25 caracteres. Digite novamente...\n");
           free(novoProduto); // Libera a memória alocada para o novo produto
           i --:
           continue;
       printf("Preço Unitário do item: ");
       scanf("%f", &novoProduto->preco);
       if (novoProduto->preco < 0) // Verifica se o preço é válido (maior ou igual a zero)
           printf("\nErro!! O preco deve ser major ou igual a zero. Digite novamente...\n"):
           free(novoProduto); // Libera a memória alocada para o novo produto
           continue:
       printf("Quantidade do item em estoque: ");
       scanf("%d", &novoProduto->quantidade);
       if (novoProduto->quantidade < 0) // Verifica se a quantidade é válida (maior ou igual a zero)
           printf("\nErro!! \ A \ quantidade \ deve \ ser \ maior \ ou \ igual \ a \ zero. \ Digite \ novamente...\n");
           free(novoProduto); // Libera a memória alocada para o novo produto
           i--;
           continue;
        if (*estoque == NULL) // Se o estoque estiver vazio, insere o novo produto como o primeiro
           *estoque = novoProduto;
       else
           Produto* atual = *estoque;
           while (atual->proximo != NULL) // Encontra o último produto no estoque
               atual = atual->proximo;
           atual->proximo = novoProduto; // Insere o novo produto no final do estoque
       printf("Produto cadastrado com sucesso!\n");
^{\prime}
void atualizarProduto(Produto* estoque) // Função para atualizar um produto do estoque
    exibirEstoque(estoque); // Chama a função exibirEstoque para mostrar o estoque atual
   if (estoque == NULL) // Verifica se o estoque está vazio
       printf("\nEstoque vazio\n");
       return; // Retorna da função
   unsigned long long codigo; // Variável para armazenar o código do produto a ser atualizado
   printf("\n--- Atualizar Produto ---\n");
   printf("Digite o codigo do produto a ser atualizado: ");
   scanf("%llu", &codigo); // Lê o código do produto a ser atualizado
    Produto* atual = estoque; // Ponteiro para percorrer a lista de produtos
   while (atual != NULL) // Enquanto houver produtos na lista
        if (atual->codigo == codigo) // Verifica se o código do produto atual é igual ao código digitado
        {
           printf("\nProduto encontrado!\n");
           printf("| Item (Codigo)\t| Nome do item\t\t\t| Valor (Unidade)\t| Estoque\n");
            printf("| \$llu \t| \$-25s \t| R\$ \$.2f \t| \$d \n", atual-> codigo, atual-> nome, atual-> preco, atual-> quantidade); 
           float novoPreco; // Variável para armazenar o novo preço do produto
           int novaQuantidade; // Variável para armazenar a nova quantidade do produto
           printf("Digite o novo preco unitario do produto: ");
           scanf("%f", &novoPreco); // Lê o novo preço do produto
           \verb|printf("Digite a nova quantidade do produto em estoque: ");\\
```

{

```
scanf("%d", &novaQuantidade); // Lê a nova quantidade do produto em estoque
           printf("\n Alteracao realizada!\n");
          printf("----\n");
          printf("Preco anterior: R$ %.2f\n", atual->preco);
          printf("Quantidade anterior: %d\n", atual->quantidade);
          printf("----\n");
          printf("Preco atualizado: R$ %.2f\n", novoPreco);
          printf("Quantidade atualizada: %d\n", novaQuantidade);
          printf("----\n");
           atual->preco = novoPreco; // Atualiza o preço do produto
          atual->quantidade = novaQuantidade; // Atualiza a quantidade do produto
           return; // Retorna da função após a atualização
       atual = atual->proximo; // Move para o próximo produto
   printf("\nProduto nao encontrado\n");
void excluirProduto(Produto** estoque) // Função para excluir os produtos
   if (*estoque == NULL) // Verifica se o estoque está vazio
       printf("\nNenhum produto cadastrado.\n");
       return; // Retorna da função
   Produto* aux = *estoque; // criando um ponteiro auxiliar para mostrar a lista atual
   exibirEstoque(aux); // Chama a função exibirEstoque para mostrar o estoque atual
   unsigned long long codigo; // Variável para armazenar o código do produto a ser atualizado
   printf("\nDigite o codigo do produto a ser excluido: ");
   scanf("%llu", &codigo);
   Produto* atual = *estoque; // Ponteiro para percorrer a lista de produtos
   Produto* anterior = NULL; // criando um ponteiro com valor vazio
   while (atual != NULL) // Enquanto houver produtos na lista
       if (atual->codigo == codigo) // Verifica se o código do produto atual é igual ao código fornecido
          printf("\nProduto encontrado:\n");
          printf("| Item (Codigo)\t| Nome do item\t\t\t| Valor (Unidade)\t| Estoque\n");
          printf("| \$llu\t\t| \$-25s\t| R\$ \$.2f\t\t| \$d\n", atual->codigo, atual->nome, atual->preco, atual->quantidade);
          printf("\nDeseja realmente excluir o produto? (S/N): ");
          char resposta:
           scanf(" %c", &resposta);
           if (resposta == 'S' || resposta == 's') // Se a resposta for 'S' ou 's', o produto será excluído
              if (anterior == NULL) // Se o produto a ser excluído for o primeiro da lista
                  *estoque = atual->proximo; // O ponteiro de estoque apontará para o próximo produto
              else
                  anterior->proximo = atual->proximo; // O produto anterior terá seu ponteiro próximo atualizado para pular o produto atual
              free(atual); // Libera a memória alocada para o produto atual
              printf("\nProduto excluido com sucesso.\n");
           else
              printf("\nExclusao cancelada.\n");
          return;
       anterior = atual; // Atualiza o ponteiro anterior para apontar para o produto atual
       atual = atual->proximo; // Avança para o próximo produto na lista
   printf("\nProduto nao encontrado.\n");
void salvarProdutos(Produto* estoque)
                                   // Função para salvar os produtos em um arquivo
   FILE* arquivo = fopen("produtos.txt", "w");
   if (arquivo == NULL)
       printf("Erro ao abrir o arquivo\n");
       return; // Retorna da função
   Produto* atual = estoque; // Ponteiro para percorrer a lista de produtos
```

```
while (atual != NULL) // Enquanto houver produtos na lista
               fprintf(arquivo, "%llu\n", atual->codigo); // Escreve o código do produto no arquivo
               fprintf(arquivo, "%s\n", atual->nome); // Escreve o nome do produto no arquivo
               fprintf(arquivo, "%.2f\n", atual->preco); // Escreve o preço do produto no arquivo
               fprintf(arquivo, "%d\n", atual->Vendas); // Escreve as vendas do produto no arquivo
               atual = atual->proximo; // Move para o próximo produto
                                         // Fecha o arquivo
       fclose(arquivo);
       printf("\nProdutos salvos com sucesso\n");
void lerProdutos(Produto** estoque) // Função para ler os produtos de um arquivo e carregá-los na memória
       FILE* arquivo = fopen("produtos.txt", "r"); // Abre o arquivo "produtos.txt" em modo de leitura
       if (arquivo == NULL)
                                                 // Verifica se houve erro ao abrir o arquivo
               printf("Erro ao abrir o arquivo\n");
              return; // Retorna da função
       Produto* atual = *estoque;// Ponteiro para percorrer a lista atual de produtos
       // Limpar a lista atual de produtos
       while (atual != NULL) // Enquanto houver produtos na lista
               Produto* proximo = atual->proximo; // Armazena o próximo produto
               free(atual); // Libera a memória ocupada pelo produto atual
               atual = proximo; // Move para o próximo produto
        *estoque = NULL;
                                            // Define o estoque como vazio (NULL)
       unsigned long long codigo; // Variável para armazenar o código do produto
       char nome[26]; // Variável para armazenar o nome do produto
       float preco; // Variável para armazenar o preço do produto
       int quantidade; // Variável para armazenar a quantidade do produto
       int Vendas; // Variável para armazenar as vendas do produto
       while (fscanf(arquivo, "%llu\n", &codigo) == 1) // Lê o código do produto do arquivo
               fgets(nome, sizeof(nome), arquivo); // Lê o nome do produto do arquivo
               nome[strcspn(nome, "\n")] = '\0'; // Remove o caractere de nova linha do nome
               fscanf(arquivo, "%f\n", &preco); // Lê o preço do produto do arquivo
                \begin{tabular}{ll} \hline & & & & \\ \hline & & & \\ \hline & & & \\ \hline & & \\
               fscanf(arquivo, "%d\n", &Vendas); // Lê as vendas do produto do arquivo
               Produto* novoProduto = (Produto*) malloc(sizeof(Produto)); // Aloca memória para o novo produto
               novoProduto->codigo = codigo; // Atribui o código do produto
               strcpy(novoProduto->nome, nome); // Copia o nome do produto
               novoProduto->preco = preco; // Atribui o preço do produto
               novoProduto->quantidade = quantidade; // Atribui a quantidade do produto
              novoProduto->Vendas = Vendas: // Atribui as vendas do produto
              novoProduto->proximo = NULL; // Define o próximo do novo produto como NULL
               if (*estoque == NULL) // Se o estoque estiver vazio
                       *estoque = novoProduto; // Define o estoque como o novo produto
               1
                      atual = *estoque; // Define o ponteiro atual como o início do estoque
                      while (atual->proximo != NULL) // Enquanto não chegar ao final do estoque
                             atual = atual->proximo; // Move para o próximo produto
                      atual->proximo = novoProduto; // Define o próximo do produto atual como o novo produto
       fclose(arquivo);
                                          // Fecha o arquivo
       printf("\nProdutos carregados com sucesso\n");
Produto* estoque = NULL; // Cria um ponteiro para o estoque de produtos e o inicializa como NULL
       int opc; // Variável para armazenar a opção escolhida pelo usuário
       lerProdutos(&estoque);
       do
               printf("\n--- MENU PRODUTOS ----\n");
              printf("1. Exibir\n");
               printf("2. Cadastrar\n");
              printf("3. Atualizar\n");
              printf("4. Excluir\n");
```

```
printf("6. Ler\n");
       printf("7. Voltar\n");
       printf("----\n");
       scanf("%d", &opc); // Lê a opção escolhida pelo usuário
       getchar(); // Lê o caractere de nova linha residual
       switch (opc)
       case 1:
           exibirEstoque(estoque); // Chama a função exibirEstoque passando o ponteiro para o estoque como argumento
       case 2:
           cadastrarProduto(&estoque); // Chama a função cadastrarProduto passando o endereço do ponteiro para o estoque como argumento
           hreak:
           atualizarProduto (estoque); // Chama a função atualizarProduto passando o ponteiro para o estoque como argumento
           excluirProduto(&estoque); // Chama a função excluirProduto passando o endereço do ponteiro para o estoque como argumento
           break:
           salvarProdutos (estoque); // Chama a função salvarProdutos passando o ponteiro para o estoque como argumento
       case 6:
           lerProdutos (&estoque); // Chama a função lerProdutos passando o endereco do ponteiro para o estoque como argumento
       case 7:
           printf("\nVoltando para o menu principal...\n");
           return; // Retorna da função para voltar ao menu principal
       default:
           printf("\nOpção inválida! Digite novamente.\n");
   while (opc < 1 || opc > 7 || opc != 7); // Repete o loop enquanto a opção escolhida for inválida ou diferente de 7
^{\prime}
void Realizar Venda(Produto** estoque)
   unsigned long long codigo;
   char resposta;
   int quantidade, tamanhoCupom = 0, opcPagamento, NumParcelas;;
   float total = 0; // Variável para calcular o total das vendas
   ItemCupomFiscal cupomFiscal[100]; // Vetor para armazenar os produtos do cupom fiscal
   if(*estoque == NULL)
       printf("\nNenhum produto cadastrado. Cadastre produtos antes de realizar uma venda.\n");
   do
       exibirEstoque(*estoque);
       printf("Digite o codigo do produto a ser vendido: ");
       scanf("%llu", &codigo);
       Produto* atual = *estoque;
       while(atual != NULL)
           if(atual->codigo == codigo)
               printf("\nProduto encontrado!\n");
               printf("| Item (Codigo)\t| Nome do item\t\t| Valor (Unidade)\t| Estoque\n");
               printf("| %llu\t\t| %-25s\t| R$ %.2f\t\t| %d\n", atual->codigo, atual->nome, atual->preco, atual->quantidade);
               printf("Digite a quantidade a ser vendida: ");
               scanf("%d", &quantidade);
               if(quantidade \leq 0 \mid \mid quantidade > atual->quantidade)
                   printf("\nQuantidade inválida ou insuficiente.\n");
               float subtotal = quantidade * atual->preco;
               atual->quantidade -= quantidade;
               atual->Vendas += quantidade;
               atual->Subtotal += subtotal;
               ItemCupomFiscal item;
               item.produto = atual;
               item.quantidade = quantidade;
               item.subtotal = subtotal;
```

printf("5. Salvar\n");

```
cupomFiscal[tamanhoCupom] = item:
                              tamanhoCupom++;
                              total += subtotal;
                              break;
                       atual = atual->proximo;
               if (atual == NULL)
                      printf("\nProduto nao encontrado\n");
               printf("\nDeseja realizar outra venda? (s/n): ");
               scanf(" %c", &resposta);
       while (resposta == 's' || resposta == 'S');
       Produto** CupomFiscal = (Produto**) malloc(tamanhoCupom * sizeof(Produto*)); // Criação do vetor do cupom fiscal
       if (CupomFiscal == NULL)
               printf("\nErro de alocação de memória\n");
               return:
       for(int i = 0; i < tamanhoCupom - 1; i ++)
               for(int j = 0; j < tamanhoCupom - i - 1; j++)
                       if(cupomFiscal[j].subtotal < cupomFiscal[j+1].subtotal)</pre>
                              ItemCupomFiscal temp = cupomFiscal[j];
                              cupomFiscal[j] = cupomFiscal[j+1];
                              cupomFiscal[j+1] = temp;
                       }
       printf("\nCupom Fiscal:\n");
       printf("| Item\t\t| Nome do Item\t\t| Valor Unit. (R$)\t| Quant.\t| Sub-Total (R$)\n");
       for (int i = 0; i < tamanhoCupom; i++)
               printf("| %d\t| %-20s\t| R$ %.2f\t| %d\t| R$ %.2f\n", i + 1, cupomFiscal[i].produto->nome, cupomFiscal[i].produto->preco, for the context of the context o
cupomFiscal[i].quantidade, cupomFiscal[i].subtotal);
       printf("\nTotal: R$ %.2f\n", total);
       int opc pag, opc troco, Num Parcelas;
       float Valor_total, Desconto, Acrescimo, Valor_Parcelas, Troco, Saldo;
       Valor total = total;
       printf("\nSelecione a forma de pagamento:\n1. A vista\n2. A prazo\n3. Voltar ao menu de vendas(caso nao haja compra no cupom fiscal):\n");
       do
        {
               scanf("%d", &opc pag);
              if (opc_pag < 1 || opc_pag > 3)
                      printf("\nOpcao de pagamento invalida, digite novamente\n");
       while (opc pag < 1 || opc pag > 3);
       if (opc_pag == 1)
               if (Valor total <= 50)
               {
                      Desconto = Valor total * 0.05; // se menor que R$ 50 = 5% de desconto
                      Valor total -= Desconto;
               else if (Valor_total < 100)
                       Desconto = Valor total * 0.1; // se maior que R$ 50 e menor que R$ 100 = 10\% de desconto
                      Valor_total -= Desconto;
               else
                      Desconto = Valor total * 0.18; // caso contrário, se maior que R$ 100 = 18% de desconto
                      Valor total -= Desconto;
               printf("\nPagamento a vista selecionado, precisa de troco?\n1.Sim\n2.Nao\n");
                       scanf("%d", &opc_troco);
                       if (opc troco < 1 || opc troco > 2)
```

```
printf("Opcao invalida, digite novamente: \n");
                while (opc_troco < 1 || opc_troco > 2);
                if (opc troco == 1)
                {
                       printf("Informe o valor de seu saldo: ");
                       scanf("%f", &Saldo);
                       while (Saldo < Valor total)
                               printf("Saldo insuficiente, tente novamente\n");
                               scanf("%f", &Saldo);
                       Troco = Saldo - Valor_total; // calculo do troco
                       printf("Troco: R$ %.2f\n", Troco);
                       printf("Desconto de: R$ %.2f\n", Desconto);
        else if (opc pag == 2)
               printf("\nValor total da compra: R$ %.2f\n", Valor total);
               printf("Digite o numero de parcelas: ");
               scanf("%d", &Num Parcelas);
               while (Num_Parcelas <= 0)
                       printf("Numero de parcelas invalido, digite um valor igual ou acima de 1 parcela.\n");
                       scanf("%d", &Num Parcelas);
               if (Num Parcelas > 3) // Se numero de parcelas major que 3 = 8% de acréscimo
                       Acrescimo = Valor_total * 0.08;
                       Valor total += Acrescimo;
               else
                {
                       Acrescimo = Valor total * 0.05; // Caso contrário, se for 1 ou 2 = 5% de acréscimo
                       Valor total += Acrescimo;
               Valor_Parcelas = Valor_total / Num_Parcelas;
               printf("\n%dx parcelas de: R$ %.2f\n", Num Parcelas, Valor Parcelas);
               printf("Acrescimo de: R$ %.2f\n", Acrescimo);
        printf("\n");
        free (CupomFiscal);
^{\prime}
void Relatorio_Vendas(Produto* estoque)
        if (estoque == NULL)
               printf("\nEstoque vazio\n");
                return;
        printf("| \  \, Item \  \, (Codigo) \  \, \  \, ll \  \, Nome \  \, do \  \, item \  \, \  \, tht \  \, \  \, ll \  \, \  \, ll \
        Produto* atual = estoque;
        while (atual != NULL)
               printf("| %llu \t| %-25s|t| R$ %.2f \t| %d \t| %d \n", atual->codigo, atual->nome, atual->preco, atual->quantidade, atual->Vendas);
                atual = atual->proximo;
       salvarProdutos(estoque);
void Vendas()
       Produto* estoque = NULL; // Cria um ponteiro para o estoque de produtos e o inicializa como NULL
       int opc;
       lerProdutos(&estoque);
        do
               printf("---- MENU ----\n");
               printf("1. Realizar venda\n");
               printf("2. Relatorio de vendas\n");
               printf("3. Voltar\n");
               printf("----\n");
               scanf("%d", &opc);
               getchar();
               switch (opc)
               case 1:
                       Realizar_Venda(&estoque);
```

```
case 2:
         Relatorio Vendas(estoque);
      case 3:
         printf("\nVoltando para o menu principal...\n");
         return; // Retorna da função para voltar ao menu principal
         printf("Opcao invalida, digite novamente!\n");
  while (opc < 1 || opc > 3 || opc != 3);
int main()
  int opc;
  do
     printf("---- MENU ----\n");
     printf("1. Produtos\n");
     printf("2. Vendas\n");
     printf("3. Sair\n");
     printf("----\n");
     scanf("%d", &opc);
     getchar();
     switch (opc)
     case 1:
        Produtos();
        break;
      case 2:
         Vendas();
        break;
      case 3:
        printf("\nSaindo do programa...\n");
        return 0;
     default:
         printf("\nOpção inválida! Digite novamente.\n");
  while (opc < 1 || opc > 3 || opc != 3);
```

break;