Universidade Federal de Viçosa
Departamento de Informática
INF 112 – Programação 2
Trabalho prático 2
Data de entrega: 30/11/2017

Trabalho Prático 2

Este trabalho consiste em desenvolver um programa para gerenciar um pequeno cadastro de produtos de uma loja. Ao ser executado, seu programa deverá exibir um menu similar ao apresentado na Figura 1.

Funcionalidades

Seu programa deverá possuir as seguintes funcionalidades:

1 – **Cadastrar produto**: permite cadastrar um produto (se o usuário tentar cadastrar um produto com um código já existente no sistema, o sistema não deverá cadastrá-lo).
2 – **Listar produtos**: exibe todos os produtos armazenados no sistema (note que os produtos são exibidos ordenados com base no código).
3 – **Remover produto**: remove um produto com determinado código (se não houver produto com o código fornecido no sistema, seu programa não deverá fazer nada).
4 – **Remover todos produtos**: remove todos produtos do sistema.
5 – **Consultar produto com código**: dado um código, exibe em tela o produto contendo esse código (se não houver nenhum produto com o código fornecido, o programa deverá imprimir um produto "vazio" - veja a Figura 6).
6 – **Sair**: Finaliza o sistema.

Ao inicializar o programa, ele deverá abrir um arquivo **binário** chamado "dados.dat" e carregar a partir desse arquivo os produtos já cadastrados no sistema (se esse arquivo não existir, o programa será inicializado com um "banco de dados" vazio) e, ao finalizar o sistema, ele deverá armazenar no arquivo "dados.dat" seu estado atual (ou seja, se o usuário executar o programa novamente ele deverá continuar com os mesmos produtos que haviam antes dele ter sido finalizado pela última vez). Note que, se ao finalizar o sistema o arquivo "dados.dat" não existir, ele deverá ser criado. Essa funcionalidade de salvar/carregar dados no arquivo DEVERÁ ser implementada no construtor/destrutor da classe GerenciadorProdutos.

Obs:
- TODAS AS OPÇÕES do menu deverão utilizar os mesmos códigos dos exemplos apresentados (ex: 1 para cadastrar produto).
- TODAS AS ENTRADAS/SAÍDAS deverão ocorrer na mesma ordem apresentada nos exemplos (ex: ao cadastrar um produto, primeiro deve-se ler o código, depois o nome do produto, depois o preço de custo, depois a margem de lucro e, finalmente, o imposto municipal).
- Sempre que possível reutilize funções definidas nas classes utilitárias (por exemplo, ao imprimir o preço de um produto utilize o operador << da classe Protudo).

Classes

Seu programa deverá possuir, **pelo menos**, as seguintes classes com (pelo menos) os seguintes métodos públicos:

Classe **Produto**: Representa um produto do sistema
Métodos públicos:
Produto(codigo, nome,precoCusto, margemLucro, impostoMunicipal); //Cria um produto com os dados informados
Produto: //Cria um produto com código -1, preço R$0,00, margemLucro 0, impostoMunicipal R$0,00 e nome vazio.

Dinheiro getPrecoCusto: //Retorna o preço de custo do produto
Dinheiro getPrecoVenda: //Retorna o preço de venda do produto (o preço de venda é o preço de custo, adicionado de margemLucro % e, então, somado ao impostoMunicipal)
int getCodigo: //Retorna o código do produto

const char * getNome: //Retorna o nome do produto (um array de caracteres terminado com \0)

Obs:
- O operador << deverá funcionar com sua classe! (ou seja, a impressão de “Dinheiros” deverá ser realizada com o uso do operador <<)
- A assinatura dos métodos **não** está fornecida de forma completa (ex: alguns métodos deverão ser constantes)
- Os nomes dos produtos terão, no máximo, 49 caracteres.

Classe **Dinheiro**: Representa um dinheiro (apenas valores não negativos!)
Métodos públicos:
Dinheiro(reais, centavos): //Cria um "dinheiro" com valor "reais" reais e "centavos" centavos
Dinheiro(): //Cria um "dinheiro" com valor R$ 0,00
unsigned **getReais**: //Retorna o número de reais no "dinheiro"
unsigned **getCentavos**: //Retorna o número de centavos no "dinheiro"
void **setReais**(valor): //Define o número de reais no "dinheiro"
void **setCentavos**(valor): //Define o número de centavos no "dinheiro"

Dinheiro + : //Operador de adição (soma dois “Dinheiro” e retorna, como resposta, o resultado da soma)
Dinheiro - : //Operador de subração (se o dinheiro resultante ficar negativo, ele será substituído por R$0,00)
Dinheiro **+=** : //Operador += (semântica similar à do operador += utilizado com variáveis do tipo inteiro – o objeto do lado direito deverá ser um objeto do tipo “Dinheiro”).
Dinheiro **-=** : //Operador -= (se o dinheiro resultante ficar negativo, ele será substituído por R$0,00)
Dinheiro ***** : //Operador de multiplicação (multiplica o dinheiro por um **valor real**; se o dinheiro resultante ficar negativo, ele será substituído por R$0,00)

Obs: o operador << deverá funcionar com sua classe! (ou seja, a impressão de “Dinheiros” deverá ser realizada com o uso do operador <<)
- A assinatura dos métodos **não** está fornecida de forma completa (ex: alguns métodos deverão ser constantes)

Classe **GerenciadorProdutos**: Classe responsável por armazenar e realiza operações com objetos do tipo Produto.
Métodos públicos:
GerenciadorProdutos(maxProdutos): //Cria um "GerenciadorProdutos" capaz de armazenar maxProdutos produtos

void **armazenaProduto**(p): //Armazena o produto no gerenciador (adicionando-o na posição correta de modo a manter a lista de produtos ordenada) . Se o limite de produtos que podem ser armazenado na classe for atingido, essa função deverá ignorar novas inserções.
void **removeProduto**(codigo): //Remove o produto com o código fornecido
void **removeTodosProdutos**(): //Remove todos os produtos do gerenciador
Produto **getProduto**(codigo): //Retorna o produto com o código fornecido (se não existir tal produto, a função deverá retornar um produto vazio – construído com o construtor padrão de Produto).
Produto **getIesimoProduto**(i): //Retorna o "i-esimo" produto armazenado no gerenciador (o 0-ésimo produto é o com menor código)
int **getNumProdutosCadastrados**(): //Retorna o número de produtos atualmente armazenados no gerenciador.
- A assinatura dos métodos **não** está fornecida de forma completa (ex: alguns métodos deverão ser constantes)

Note que o programa que você irá desenvolver deverá utilizar, pelo menos, essas três classes. Observe também que os métodos públicos especificados são requisitos **mínimos** das classes (elas provavelmente terão outros métodos) e, além disso, a assinatura deles está incompleta (o desenvolvimento da assinatura adequada será parte do seu trabalho).

Exemplos de tela

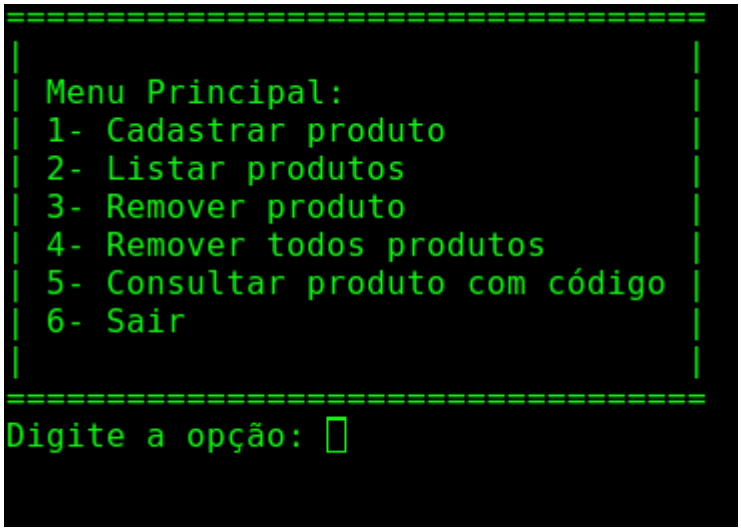

Figura 1: Menu principal.

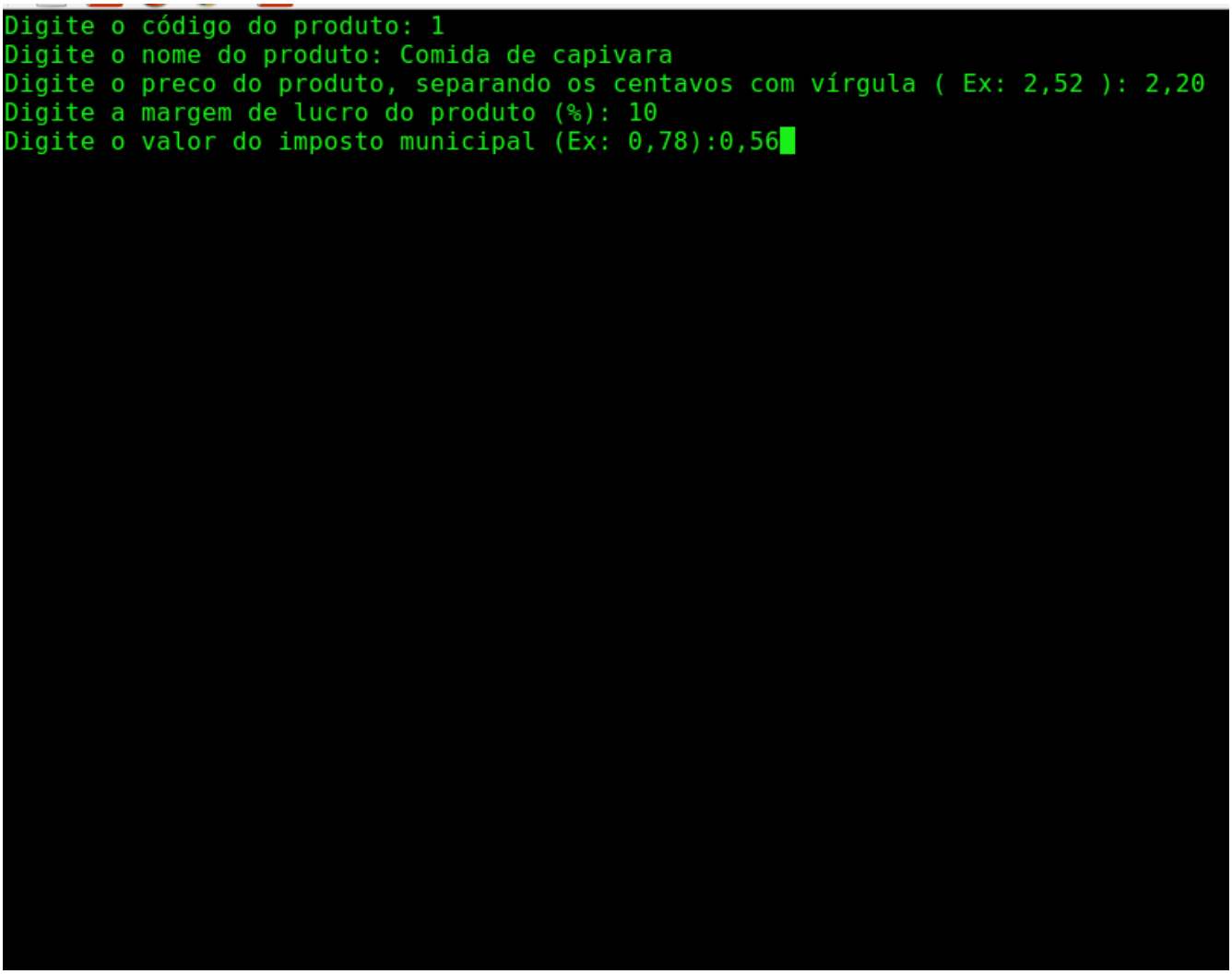

Figura 2: Cadastro de produto

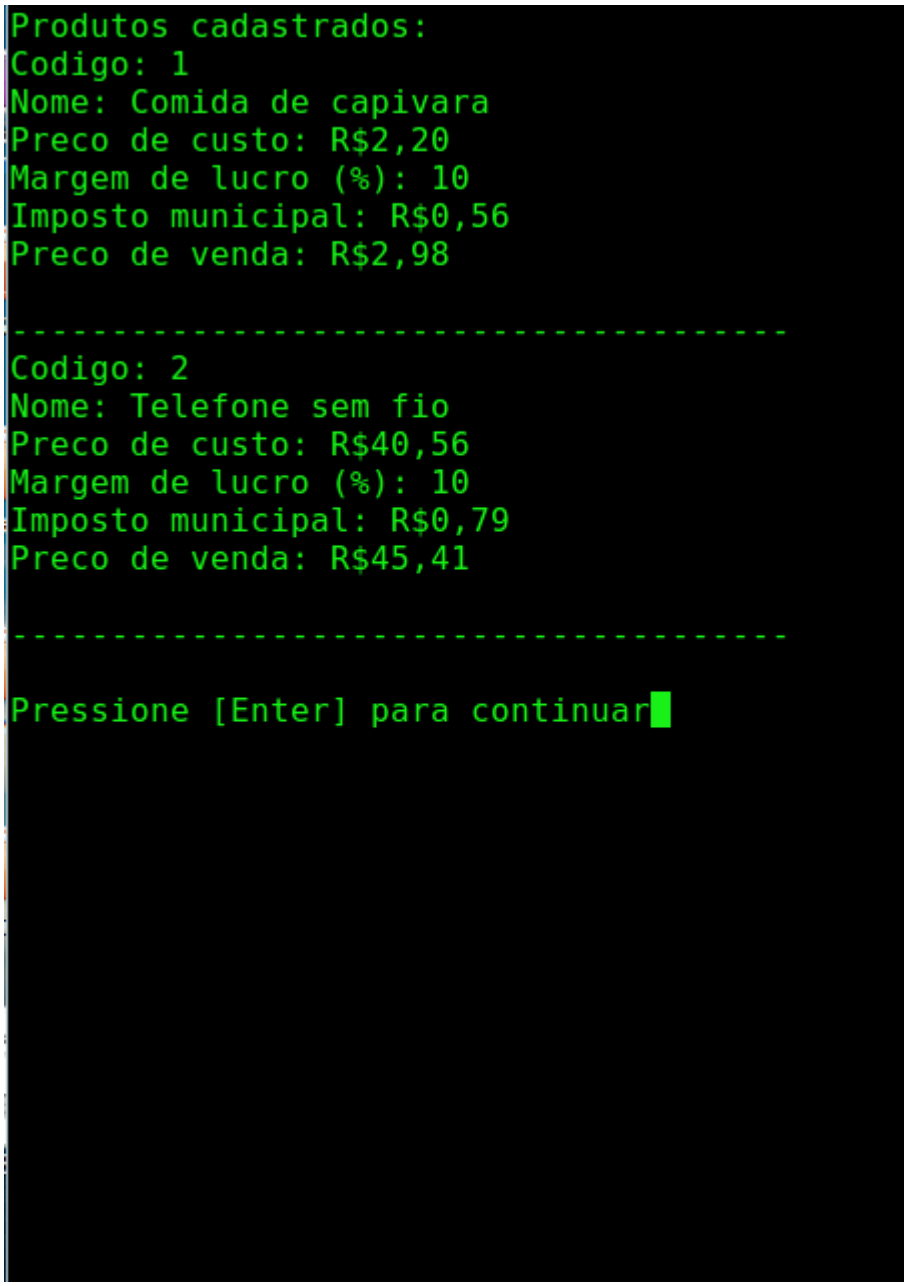

Figura 3: Listar produtos

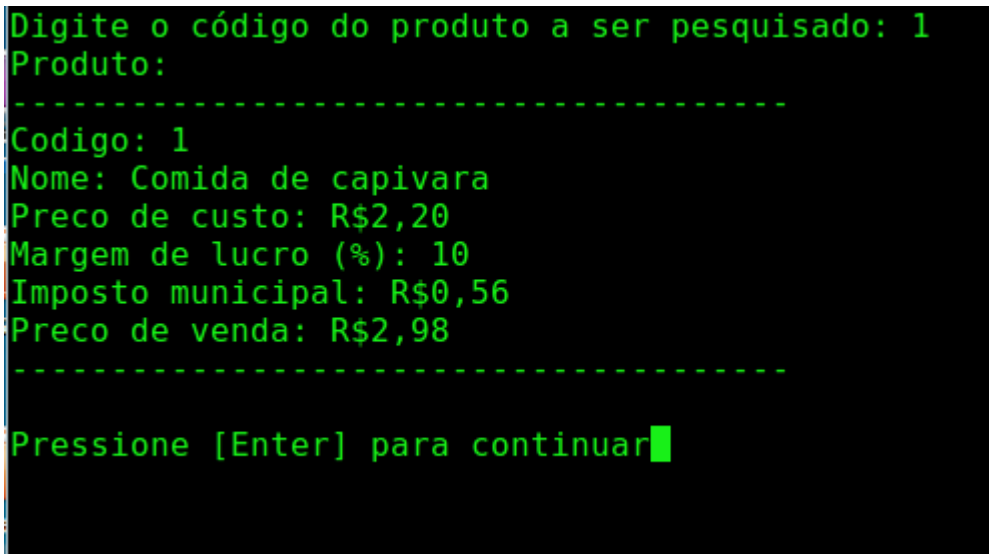

Figura 4: Consultar produto

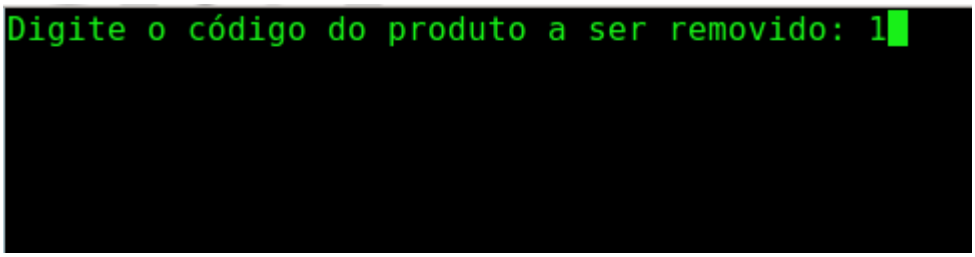

Figura 5: Remover produto

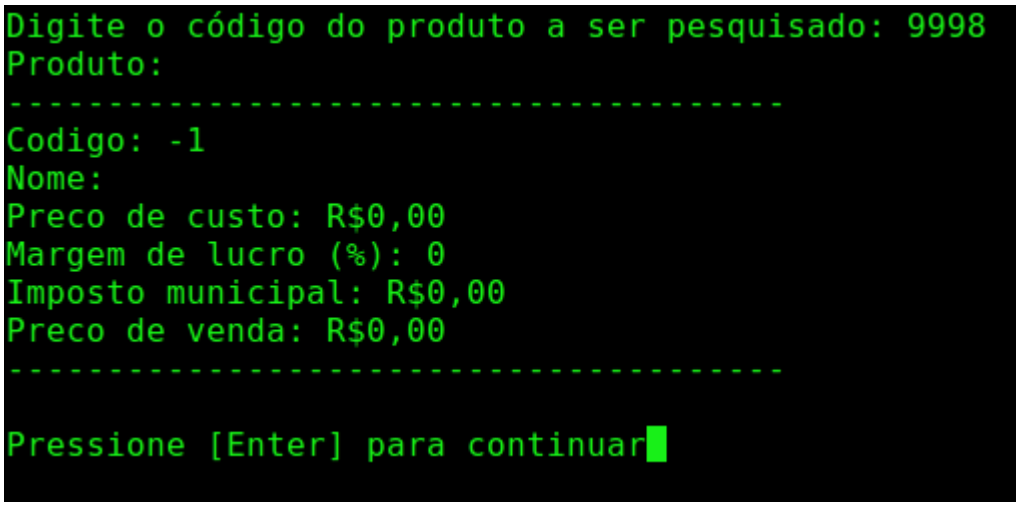

Figura 6: Consultar produto não existente

Mais observações

Entrega: devem ser entregues, através do PVAnet, o código fonte com seu respectivo Makefile e uma documentação em formato pdf com todo o memorial descritivo da solução, contendo todas as decisões de implementação e detalhes das estratégias de solução.

Na distribuição de pontos a documentação receberá, no mínimo, 50% dos pontos totais. Uma documentação de boa qualidade deverá apresentar a descrição de todas as classes propostas bem como a assinatura de todos os métodos, com descrição detalhada do que é retornado (tipo e semântica do retorno) e dos parâmetros (tipo e significado de cada um no contexto do programa).

Observe que seu trabalho deverá:
- Ser compatível com o sistema operacional Linux (ou seja, se você o desenvolver em outro Sistema Operacional você deverá testá-lo também no Linux e enviar um Makefile compatível com o Linux).
- Ter as classes/interfaces organizadas em arquivos diferentes (o Makefile deverá compilar cada um desses arquivos de forma individual e, finalmente, linká-los para gerar o executável).
- Apresentar telas similares às dos exemplos.
- Estar bem indentado, organizado e apresentando comentários explicativos no código fonte.