Universidade Federal de Viçosa
Departamento de Informática
DPI

Wallace Ferancini Rosa – 92545

TP2
Documentação da implementação

Viçosa
2017

Sumário

1. Introdução

Este documento possuí o objetivo de descrever a lógica e implementação utilizadas no desenvolvimento das interfaces, classes, métodos do Trabalho Prático 2, no qual foi solicitado o desenvolvimento de um pequeno sistema que gerencie um cadastro de produtos utilizando metodologias da Programação Orientada a Objetos de forma que o sistema possua as seguintes funcionalidades:

- **Cadastrar produto:** permite cadastrar um produto (se o usuário tentar cadastrar um produto com um código já existente no sistema, o sistema não o cadastrará).
- **Listar produtos:** exibe todos os produtos armazenados no sistema (os produtos são exibidos ordenados com base no código).
- **Remover produto:** remove um produto com determinado código (se não houver produto com o código fornecido no sistema, seu programa não deverá fazer nada).
- **Remover todos produtos:** remove todos produtos do sistema.
- **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").
- **Sair:** Finaliza o sistema.

2. Descrição das classes e métodos utilizados

O sistema implementado utiliza três classes: Dinheiro, Produto e GerenciadorProdutos. Cada uma dessas classes é, em certo nível, uma abstração da realidade: a classe Dinheiro representa valores em moeda(R$) armazenados, a classe Produto representa os produtos armazenados no sistema e a classe GerenciadorProdutos é o "manipulador" dos dados, a classe que gerencia e manipula os cadastrados(semelhante a um gerente).

2.1 Classe Dinheiro

A classe *Dinheiro* representa valores monetários, na moeda real, a serem armazenados e manipulados no sistema. Segue abaixo a descrição da classe implementada e a assinatura de seus métodos:

```
#include <iostream>

using namespace std;

class Dinheiro
```

```
{
        private:
                unsigned int Reais;
                unsigned int Centavos;
        public:
                Dinheiro(unsigned int Reais,unsigned int Centavos); Dinheiro();
                ~Dinheiro();
                unsigned getReais() const;
                unsigned getCentavos() const;
                void setReais(unsigned int Reais);
                void setCentavos(unsigned int Centavos);
                Dinheiro & operator=(const Dinheiro d2);
                Dinheiro & operator+=(const Dinheiro d2);
                Dinheiro & operator-=(const Dinheiro d2);
                Dinheiro operator+(const Dinheiro d2) const;
                Dinheiro operator-(const Dinheiro d2) const;
                Dinheiro operator*(const double c) const;


                friend ostream & operator<<(ostream & os, const Dinheiro & d);
};
```

Membros privados da classe

- **unsigned int Reais, unsigned int Centavos:** são os membros da classe que representam o valor monetário, em reais, propriamente dito onde *Reais* é o membro representante das unidades de real e o membro *Centavos* as unidades de centavos informadas. Como está contido na descrição e não faz sentido neste contexto armazenarmos valores monetários negativos utiliza-se o tipo unsigned int para indicar valores inteiros que não marcação de sinal, ou seja, representam valores positivos.

Membros públicos

- **Dinheiro(unsigned int Reais, unsigned int Centavos):** construtor com parâmetros da classe *Dinheiro*. Inicia o objeto instanciado com os valores dos parâmetros recebidos. O membro *Reais* recebe o valor do parâmetro *unsigned int Reais* e o membro *Centavos* recebe o valor do parâmetro *unsigned int Centavos*

- **Dinheiro():** construtor sem parâmetros. Inicia o objeto isntanciado com o valor zero(“*Reais = 0”* e *“Centavos= 0”*).
- **~Dinheiro():** destrutor da classe *Dinheiro*. Não realiza nenhuma operação relevante dentro de si mesmo.
- **unsigned getReais() const:** método *get* relacionado ao membro *Reais.* Retorna o valor inteiro de *Reais.* Método constante, não altera nenhum dado da instancia do objeto que chama o método.
- **unsigned getCentavos() const:** análogo ao método *getReais()* porém retorna o valor de Centavos.
- **void setReais(unsigned int Reais):** método *set* relacionado ao membro *Reais*. Altera o valor inteiro de *Reais* da instancia do objeto pelo valor inteiro do parâmetro *unsigned int Reais.*
- **void setCentavos(unsigned int Centavos):** análogo ao método *setReais()* porém altera o valor do membro *Centavos* da instancia do objeto.
- **Dinheiro & operator=(const Dinheiro d2):** sobrecarga do operador de atribuição “=” na classe Dinheiro. Realiza atribuição de uma instancia de *const Dinheiro d2(* parâmetro do método*)* a outra instancia de *Dinheiro* (a instância que “chama” o método, ex: “d = d2” sendo d a instância que executa a operação), copiando os dados de *Reais* e *Centavos* de d2 para a instância que “chama” o método.
- **Dinheiro operator+(const Dinheiro d2) const:** sobrecarga do operador de soma “+” na classe Dinheiro. Realiza a soma de um *Dinheiro*(a instância que “chama” o método, ex: “d + d2” sendo d a instância que executa a operação) a outro const *Dinheiro* d2*,* parâmetro do método. Primeiramente os valores de *Reais* e *Centavos* das instâncias são convertidos para o devido valor em centavos. Após isso é realizada a soma dos valores, e, então, é retornado outro *Dinheiro* que representa o valor da soma, extraindo o devido valor em reais da soma, atribuindo-o a *Reais,* e atribuindo o restante do valor a *Centavos*. O método é constante, pois não altera nenhum dos dados de d2 ou da instância que executa o método, apenas retornando um *Dinheiro* que representa a soma.
- **Dinheiro & operator+=(const Dinheiro d2):** sobrecarga do operador de soma “+=” na classe Dinheiro. Realiza a soma de um *Dinheiro* (a instância que “chama” o método, ex: “d += d2” sendo d a instância que executa a operação) a outro *Dinheiro* d2(parâmetro do método) e atribui o valor da soma à instância que executa o método. Primeiramente os valores de *Reais* e *Centavos* das instâncias são convertidos para o devido valor em centavos. Após isso é realizada a soma dos valores , a qual é então atribuida à instância que executa o método, extraindo o devido valor de reais da

soma e atribuindo a *Reais* e atribuindo o restante a *Centavos.* Após realizar a atribuição o método retorna uma referência da instância que o executou.

- **Dinheiro operator-(const Dinheiro d2) const:**sobrecarga do operador de soma "-" na classe Dinheiro. Realiza a subtração de um *Dinheiro*(a instância que "chama" o método, ex: "d - d2" sendo d a instância que executa a operação) por outro *Dinheiro* d2*,* parâmetro do método. Primeiramente os valores de *Reais* e *Centavos* das instâncias são convertidos para o devido valor em centavos. Após isso é realizada a subtração dos valores, e, então, é retornado outro *Dinheiro* que representa o valor da subtração, extraindo o devido valor em reais da soma, atribuindo-o a *Reais,* e atribuindo o restante do valor a *Centavos*. O método é constante, pois não altera nenhum dos dados de d2 ou da instância que executa o método, apenas retornando um *Dinheiro* que representa a subtração.
- **Dinheiro & operator-=(const Dinheiro d2):** sobrecarga do operador de soma "-=" na classe Dinheiro. Realiza a subtração de um *Dinheiro* (a instância que "chama" o método, ex: "d -= d2" sendo d a instância que executa a operação) por outro *Dinheiro* d2(parâmetro do método) e atribui o valor da subtração à instância que executa o método. Primeiramente os valores de *Reais* e *Centavos* das instâncias são convertidos para o devido valor em centavos. Após isso é realizada a subtração dos valores , a qual é então atribuida à instância que executa o método, extraindo o devido valor de reais da soma e atribuindo a *Reais* e atribuindo o restante a *Centavos.* Após realizar a atribuição o método retorna uma referência da instância que o executou.
- **Dinheiro operator*(const double c) const:** sobrecarga do operador de soma "*" na classe Dinheiro. Realiza a multiplicação de um Dinheiro(a instância que "chama" o método, ex: "d * c" sendo d a instância que executa a operação) por uma constante real *const double c,* parâmetros do método. Primeiramente o valor de Reai e Centavo da instância é convertido para o devido valor em centavos. Após isso é realizada a mutiplicação do valor pela constante, e, então, é retornado outro Dinheiro que representa o valor da multiplicação, extraindo o devido valor em reais da soma, atribuindo-o a Reais, e atribuindo o restante do valor a Centavos. O método é constante, pois não altera nenhum dos dados de d2 ou da instância que executa o método, apenas retornando um Dinheiro que representa a multiplicação.
- **friend ostream & operator<<(ostream & os, const Dinheiro & d):** sobrecarga do operador de "<<", que neste caso é usado para impressão de uma saída pelo comando *cout*. O método captura a referência da classe *ostream* do *cout* que realiza a

impressão do *Dinheiro d* a ser impresso na tela e então é utilizado o operador "<<" de *ostream* para retornar os valores de *Reais* e *Centavos* do *Dinheiro* em questão. O método então retorna a referência a instância de *ostream* em questão, que ira então realizar a impressão dos dados na tela.

2.2 Classe Produto

A classe *Produto* representa produtos, que possuem codigo, nome, preço de custo, margem de lucro e valor de imposto municipal associados, a serem armazenados e manipulados no sistema. Segue abaixo a descrição da classe implementada e a assinatura de seus métodos:

```
#include "Dinheiro.h"
#include <cstring>
#include <iostream>
class Produto
{
	private:
		int Codigo;
		char * Nome;
		Dinheiro PrecoCusto;
		double MargemLucro;
		Dinheiro ImpostoMunicipal;
	public:
		Produto(int Codigo, char * Nome, Dinheiro PrecoCusto, double MargemLucro, Dinheiro ImpostoMunicipal);
		Dinheiro ImpostoMunicipal);
		Produto();
		~Produto();
		Produto(Produto &p);
		Dinheiro getPrecoCusto() const;
		Dinheiro getPrecoVenda() const;
		Dinheiro getImpostoMunicipal() const;
		int getCodigo() const;
		const char * getNome() const;
		double getMargemLucro() const;
```

```
        Produto & operator=(const Produto p2);


        friend ostream & operator<<(ostream & os, const Produto & p);
};
```

Membros privados da classe

- **int Codigo :** representa o código do produto a ser armazenado no sistema. Códigos com número positivo representam produtos válidos, enquanto -1 representa um produto positivo
- **char * Nome :** representa o nome do produto a ser armazenado no sistema. Os nomes dos produtos possuem no máximo 49 caracteres.
- **Dinheiro PrecoCusto :** membro que representa o preco de custo(preço necessário para comprar o produto de algum fornecedor). Valor monetário definido pelo tipo *Dinheiro*.
- **double MargemLucro :** membro que representa a margem de lucro a ser atingida pelo produto em sua venda. Valor monetário definido pelo tipo *Dinheiro*.
- **Dinheiro ImpostoMunicipal :** membro que representa o valor do tributo municipal sobre o produto. Valor monetário definido pelo tipo *Dinheiro*.

Membros públicos

- **Produto(int Codigo, char * Nome, Dinheiro PrecoCusto, double MargemLucro, Dinheiro ImpostoMunicipal) :** Construtor com parâmetros da classe *Produto.* Cria uma instância de *Produto* e a inicia os membros privados com os valores dos parâmetros(*Codigo* recebe o valor do parâmetro *Codigo*, *Nome* recebe o conteúdo do parâmetro *Nome*(*Nome* é alocado dinamicamente como um vetor de *char* com 50 posições, 49 caracteres e o '\0'), *PrecoCusto* recebe o parâmetro *PrecoCusto*(membros de *PrecoCusto* recebem os valores dos membros do parâmetro), *MargemLucro* recebe o valor do parâmetro *MargemLucro* e *PrecoCusto* é iniciado com o parâmetro *ImpostoMunicipal*.
- **Produto() :** Construtor sem parâmetros da classe *Produto*. Cria uma instância de *Produto* "vazia", *Codigo* recebe -1, *Nome* é iniciado com " "(*Nome* é alocado dinamicamente como um vetor de *char* com 50 posições, 49 caracteres e o '\0'), *PrecoCusto* recebe um *Dinheiro* que representa R$0,00, *MargemLucro* recebe 0.00 e *PrecoCusto* é iniciado com um *Dinheiro* que representa R$0,00.
- **~Produto() :** Destrutor da classe *Produto.* Ao ser chamado, libera a memória de *Nome* que foi alocada dinamicamente.
- **Produto(Produto &p)(criar Produto(const Produto &p para classes) :** Construtor de cópia da classe *Dinheiro.* Realiza uma cópia dos valores do parâmetro *p* para a instância de *Produto* que chama o método de cópia.

- **Dinheiro getPrecoCusto() const :** método *get* que retorna o *Dinheiro* armazenado por *PrecoCusto.* Método constante, pois não altera a instância que executa o método.
- **Dinheiro getPrecoVenda() const :** método *get* que retorna um *Dinheiro* que representa o valor de venda do produto(Preço de custo multiplicado pela margem de lucro). É criado um *Dinheiro* que recebe o valor de *PrecoCusto* multiplicado por *MargemLucro* fazendo uso da sobrecarga do operador "*". Esse *Dinheiro*, que representa o valor de venda do produto, é então retornado pelo método. Método constante, pois não altera a instância que executa o método.
- **Dinheiro getImpostoMunicipal() const :** método *get* que retorna o *Dinheiro* amazenado por *ImpostoMunicipal.* Método constante, pois apenas retorna uma instância de *Dinheiro* sem alterar a instância que executa o método.
- **int getCodigo() const :** método *get* que retorna o inteiro que representa o código de um produto, armazenado por *Codigo.* Método constante, pois não altera a instância que executa o método.
- **const char * getNome() const :** método *get* que retorna o um ponteiro de char constante com o nome de um produto, armazenado por *Nome.* Método constante, pois não altera a instância que executa o método.
- **double getMargemLucro() const :** método *get* que retorna o um double que representa a margem de lucro de algum produto, armazenado por *MargemLucro.* Método constante, pois não altera a instância que executa o método.
- **Produto & operator=(const Produto p2) :** sobrecarga do operador de atribuição "=" na classe *Produto.* Realiza uma cópia dos dados armazenados de p2 para a instância de *Produto* que executa o método(ex : "p = p2" , sendo "p" a instância que executa o método). O parâmetro *p2* é constante pois nenhum de seus dados serão alterados. O método retorna uma referência da instância que executa o método.
- **friend ostream & operator<<(ostream & os, const Produto & p) :** sobrecarga do operador de "<<", que neste caso é usado para impressão de uma saída pelo comando *cout*. O método captura a referência da classe *ostream* do *cout* que realiza a impressão do *Produto p* a ser impresso na tela e então é utilizado o operador "<<" de *ostream* para retornar à saída os valores de *p*(código, nome, preço de custo, preço de venda e imposto municipal) em questão. O método então retorna a referência a instância de *ostream* em questão, que ira então realizar a impressão dos dados na tela.

2.3 Classe GerenciadorProdutos

A classe GerenciadorProdutos é o "operário" do sistema. É a classe responsável por armazenar e gerenciar os produtos. A classe basicamente possuí um vetor de *Produto* alocado dinamicamente, com um número máximo de produtos a ser determinado pelo usuário e um contador de produtos cadastrados. Essa classe também manipula o armazenamento dos dados utilizando um

arquivo binário “dados.dat”(ao iniciar o programa os produtos armazenados no arquivo são carregados e ao finalizar o programa os produtos cadastrados são armazenados no arquivo). Segue abaixo a descrição da classe implementada e a assinatura de seus métodos:

```
#include "Produto.h"
#include <fstream>

class GerenciadorProdutos
{
	private:
		int MaxProdutos;
		int ProdutosCadastrados;
		Produto * Lista;
	public:
		GerenciadorProdutos(int MaxProdutos);
		GerenciadorProdutos(GerenciadorProdutos & g);
		~GerenciadorProdutos();
		void armazenaProduto(Produto &p);
		void removeProduto(int codigo);
		void removeTodosProdutos();
		Produto getProduto(int codigo) const;
		Produto getIesimoProduto(int i) const;
		int getNumProdutosCadastrados() const;
		void leProdutoDoTeclado();
		void listarProdutos() const;

		friend int pesquisaBinaria(Produto * p, int inicio, int fim, int chave);

};
```

Membros privados da classe

- **int MaxProdutos :** representa o número máximo de produtos a serem armazenados no sistema.
- **int ProdutosCadastrados :** representa o número de produtos cadastrados no sistema durante a execução.
- **Produto * Lista :** representa os produtos cadastrados no sistema. Neste caso, é um vetor de *Produto* alocado dinamicamente.

Membros públicos

- **GerenciadorProdutos(int MaxProdutos) :** Construtor com parâmetros da classe *GerenciadorProdutos.* Aloca um vetor de *Produto*(*Lista*) dinamicamente, com numero de posições informado pelo parâmetro MaxProdutos e inicia *ProdutosCadastrados* com o valor 0. Após isso, é checado se o arquivo de entrada “dados.dat” existe. Caso o arquivo exista, os registros de produtos que estiverem armazenado nele serão carregados no vetor *Lista.*
- **GerenciadorProdutos(GerenciadorProdutos & g) :** Construtor de cópia da classe *Dinheiro.* Realiza uma cópia dos valores do parâmetro *g* para a instância de *GerenciadorProdutos* que chama o método de cópia(Os dados do vetor *Lista*, o valor de *MaxProdutos*, e o valor de *ProdutosCadastrados*).

- **~GerenciadorProdutos() :** Destrutor da classe *GerenciadorProdutos.* Desaloca a memória alocada dinamicamente para o vetor *Lista.* Além disso, armazena no arquivo binário "dados.dat" os dados dos produtos cadastrados durante a execução do programa.
- **friend int pesquisaBinaria(Produto * p, int inicio, int fim, int chave) :** método que executa uma busca binária(de modo recursivo) em um vetor de Produtos *p* alocado dinamicamente. O método recebe como parâmetros um vetor de *Produto*, *p,* uma faixa de busca(posição de início da busca marcada por *inicio* e a posição do fim da busca marcada por *fim*) e o código do produto que se deseja encontrar(marcado por *chave*). Caso o produto seja encontrado na busca, é retornado a posição do produto no vetor *p.* Caso o produto não seja encontrado, é retornado o valor -1.
- **void armazenaProduto(Produto &p) :** método que armazena um produto no sistema. Recebe como parâmetro o *Produto p* a ser armazenado e o insere no vetor *Lista,* aumentando o contador de produtos cadastrados(*ProdutosCadastrados*) em um. Caso o número de produtos cadastrados já tenha atingido o limite, a inserção do produto é ignorada e a operação é abortada, nada ocorre. É realizada uma pesquisa no vetor *Lista* utilizando o método *pesquisaBinaria*, e, caso o produto já exista no sistema a inserção é abortada e nada ocorre. Ao realizar a inserção de um *Produto* o resto do vetor é reorganizado sequencialmente a partir da posição em questão de forma a permanecer ordenado: é aplicado um "backspace" naquela posição, "arredando" o restante do vetor.
- **void removeProduto(int codigo) :** método que remove um produto do vetor *Lista.* É realizada uma pesquisa no vetor *Lista* utilizando o método *pesquisaBinaria,* pelo qual é obtida a posição do vetor em que o *Produto* em questão se encontra, e então o *Produto* é excluído do vetor sendo substituído por um *Produto* "vazio". Ao realizar a substituição o resto do vetor é reorganizado sequencialmente a partir da posição em questão de forma a permanecer ordenado em ordem crescente em relação ao valor de *Codigo*: é aplicado um "backspace" naquela posição, "arredando" o restante do vetor.
- **void removeTodosProdutos() :** método que remove todos os produtos cadastrados no sistema. Basicamente o vetor *Lista* é desalocado e o contador *ProdutosCadastrados* recebe o valor 0. Após isso realocamos novamente o vetor *Lista* com o devido valor de posições armazenado em *MaxProdutos,* o que garante que todos os produtos anteriormente cadastrados foram excluidos (ao alocar o vetor novamente, é chamado o construtor sem parâmetros da classe *Produto*, que alocará produtos "vazios" em todo o vetor).
- **Produto getProduto(int codigo) const :** método que retorna um *Produto* com *Codigo* igual ao parâmetro *codigo* recebido pelo método. É realizada uma busca no vetor *Lista* utilizando o método *pesquisaBinaria,* o qual retorna a posição do vetor em que o produto desejado se encontra, e então é retornado o *Produto* procurado. Caso o *Produto* desejado não exista no vetor, é retornado um *Produto* "vazio"(gerado pelo construtor sem parâmetros da classe *Produto*). O método é constante pois não altera os dados armazenados pela instância que executa o método.
- **Produto getIesimoProduto(int i) const :** método que retorna o i-ésimo produto armazenado no sistema(sendo o 0-ésimo produto o produto de menor código, e assim por diante). O parâmetro *i* recebido representa o i-ésimo produto a ser buscado. Como o vetor *Lista* é ordenado em ordem crescente em relação a *Codigo*, o i-ésimo produto é justamente o produto armazenado na posição i do vetor *Lista.* Caso o produto não exista no vetor é exibida uma mensagem de erro ao usuário e um

Produto “vazio”(gerado pelo construtor sem parâmetros da classe *Produto*) é retornado. O método é constante pois não altera os dados armazenados pela instância que executa o método.

- **int getNumProdutosCadastrados() const :** método que retorna o número de produtos cadastrados no sistema(inteiro), armazenado por *ProdutosCadastrados*. O método é constante pois não altera os dados armazenados pela instância que executa o método.
- **void leProdutoDoTeclado() :** método que cria um lê um *Produto* a partir de dados digitados pelo usuário do sistema no teclado. O método recebe os dados digitados pelo usuário(código, nome, etc), testa os dados para verificar a validade(se o código é um inteiro positivo, se os valores numéricos digitados são positivos, etc) dos dados, cria um *Produto* utilizando esses dados como parâmetros para o construtor com parâmetros da classe *Produto* e então insere o produto no sistema utilizando o método *armazenaProduto.*
- **void listarProdutos() const :** método que lista na tela de saída todos os produtos cadastrados pelo sistema. Como o vetor *Lista* é sempre ordenado em ordem crescente com relação ao código, os produtos são exibidos em ordem sequencial (começando da posição 0 até a posição *ProdutosCadastrados* – 1). O método é constante pois não altera os dados armazenados pela instância que executa o método.
- **friend int pesquisaBinaria(Produto * p, int inicio, int fim, int chave) :** método de pesquisa binária, pesquisa um *Produto* com *Codigo* desejado em um vetor e retorna a posição do vetor em que esse *Produto* se encontra, sendo os parâmetros: *p* o vetor de *Produto* no qual a pesquisa será realizada, *inicio* a posição a ser considerada inicial no vetor, *fim* a posição a ser considerada final no vetor, *chave* o valor de *Codigo* do *Produto* a ser pesquisado. Caso o produto não seja encontrado é retornado o valor -1.

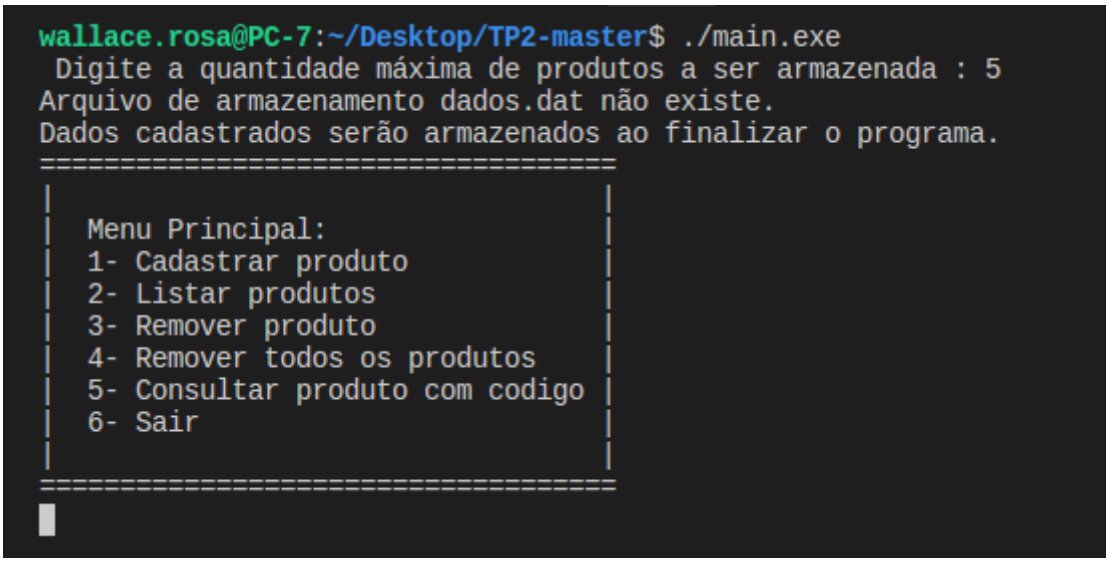
3. Sistema implementado

Para compilar o sistema compilado utilizando o sistema Linux, basta executar o makefile enviado.

O sistema , ao ser executado, perguntado ao usuário a quantidade máxima de produtos a serem cadastrados durante a execução e então checa se existe o arquivo “dados.dat”, caso ele exista os dados armazenados serão carregados. Se o arquivo existir e o usuário informar uma quantidade máxima de produtos que seja menor que o número de produtos armzenados em “dados.dat”, o sistema perguntará ao usuário se o mesmo deseja carregar todos os produtos armazenados, e, caso o usuário opte por essa opção o sistema perguntará se o usuário deseja aumentar o número de produtos a ser armazenado. Este inicio de execução é ilustrado abaixo:

```
wallace.rosa@PC-7:~/Desktop/TP2-master$ ./main.exe
 Digite a quantidade máxima de produtos a ser armazenada : 1
Arquivo de banco de dados existente: dados.dat
Buscando produtos já cadastrados...
Ha 2 produtos armazenados mas foi solicitado um numero maximo de 1 produto(s)
Deseja carregar todos os produtos ou continuar com 1 prouduto(s)?(os produtos excedentes serão excluidos)
(s = sim/ n = nao)
s
Deseja aumentar a quantidade de produtos armazenados?(Se sim informe o numero de produtos a aumentar, se nao digite um numer
o menor ou igual a zero) :
2
2 produtos foram carregados.
```

Após isso é criado uma instância de *GerenciadorProdutos,* que irá gerenciador todo fluxo de dados do sistema, com o devido número máximo de produtos informado pelo usuário e a execução do sistema consiste em um menu com as funcionalidades aboradadas na **Introdução** deste trabalho, conforme ilustrado abaixo :



Nesta parte é esperado que o usuário digite a opção da funcionalidade desejada, e então o programa executará a devida operação. Caso uma opção inválida seja informada , é exibida uma mensagem de erro e o programa retorna ao menu principal.

3.1 Opções do sistema

- **Cadastrar produto:** permite cadastrar um produto (se o usuário tentar cadastrar um produto com um código já existente no sistema, o sistema não o cadastrará). Utiliza o método *leProdutoDoTeclado* da classe *GerenciadorProdutos.* Caso algum dado inválido seja informado a operação é abortada e o programa retorna ao menu principal.
- **Listar produtos:** exibe todos os produtos armazenados no sistema (os produtos são exibidos ordenados com base no código). Utiliza o método *listarProdutos()* da classe *GerenciadorProdutos.*
- **Remover produto:** remove um produto com determinado código (se não houver produto com o código fornecido no sistema, seu programa não deverá fazer nada). Utiliza o método *removeProduto* da classe *GerenciadorProdutos.*
- **Remover todos produtos:** remove todos produtos do sistema. Utiliza o método *removeTodosProdutos* da classe *GerenciadorProdutos.*
- **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”). Utiliza o método *getProduto* da classe *GerenciadorProdutos.*

- **Sair:** Finaliza o sistema. Todos os dados salvos pela instância de *GerenciadorProdutos* serão salvos no arquivo binário “dados.dat” ao ser chamado o destrutor da classe *GerenciadorProdutos.*