# Software de Gerenciamento de Estoque de Mercearia

Giordana Bucci
*Estudante de Ciência da Computação*
*Instituto de Informática - Universidade Federal de Goiás*
*Goiânia, Goiás, Brasil (62) 3521-1181*
*Email: giordanabucci@discente.ufg.br*

Guilherme Tavares
*Estudante de Ciência da Computação*
*Instituto de Informática - Universidade Federal de Goiás*
*Goiânia, Goiás, Brasil (62) 3521-1181*
*Email: guilhermetavares@discente.ufg.br*

Pedro Lobo
*Estudante de Ciência da Computação*
*Instituto de Informática - Universidade Federal de Goiás*
*Goiânia, Goiás, Brasil (62) 3521-1181*
*Email: pedro_lemes@discente.ufg.br*

Ana Luísa Chagas
*Estudante de Ciência da Computação*
*Instituto de Informática - Universidade Federal de Goiás*
*Goiânia, Goiás, Brasil (62) 3521-1181*
*Email: analuisa23@discente.ufg.br*

***Resumo*—This report aims to detail the process of developing a software for managing grocery stores and emporiums, with the intention of applying the knowledge acquired during the course of the subject Algorithms and Data Structures 1. Several data storage structures were used, such as, sorting, and searching. The result is a software that allows better business organization and increases trade profitability.**

***Keywords: Inventory, Management, Software, Supermarket, Products*.**

## 1. Introdução

A proposta deste trabalho foi fazer um sistema de aplicação real utizando os conhecimentos adquiridos nas aulas de Algoritmo e Estrutura de Dados I e apresentá-lo em sala de aula. Com isso em mente, a solução pensada pelo grupo para o seminário consiste em um software para gerenciamento de estoque de mercearia ou empório, após a observação da quantidade de comércios que não possuem ferramentas necessárias para facilitar a organizalçao do estoque do estabelecimento.

Esse sistema foi feito com estruturas condicionais, estruturas de repetição, alocação dinâmica de memória, ponteiros, vetores, moularização, tipos abstratos de dados (TAD), listas duplamente encadeadas e métodos de ordenação - conteúdos aprendidos e aprofundados durante as aula práticas e teóricas - em sua criação. Além disso, esse gerenciador possui um menu interativo e intuitivo no começo do programa com as operações a serem realizadas, no qual cada número corresponde a uma operação. Esse menu tem funções primordiais para controle de estoque, como adicionar itens, remover itens, buscar por itens no estoque, mostrar o estoque ordenado por preço e calcular o valor total de uma compra com os itens disponíveis em estoque.

Este relatório é dividido em 5 seções. Na seção 2, será explicado cada estrutura utilizada na programação desse sistema. A seção 3 falará sobre a metodologia utilizada para o desenvolvimento dese gerenciador, isto é, como utilizamos cada estrutura da programação para fazer com que o sistema fizesse cada ação. Na seção 4, será abordado os resultados e discussão sobre o trabalho. Por fim, na seção 5 será exposta a conclusão referente à este trabalho.

## 2. Fundamentos Teóricos

Na Introdução, foi mencionado quais estruturas da programação foram utilizadas para a solução encontrada pela equipa para o trabalho proposto. Nesse sentido, discutiremos e explicaremos de maneira simples e intuitiva cada uma delas. A maior fonte de pesquisa a respeito dos conteúdos abordados foram as aulas de AED1, seus materiais disponibilizados na plataforama SIGAA-UFG além do livro Estruturas de Dados Usando C [Tenenbaum et *al*. (2004)].

As estruturas condicionais são estruturas de controle de fluxo que permitem que o programa execute diferentes ações e mude seu comportamento durante sua execução com base nas condições abordadas. As estruturas condicionais são if(), else if(), else e switch case.

As estruturas de repetição são estruturas que permitem a execução de uma parte do código mais de uma vez, até que o critério de parada seja obtido. O trecho do código dentro da função é executado quantas vezes for necessário, e as estruturas usadas são: while, for e do while.

A alocação dinâmica de memória: Gerenciamento manual de memória por meio de funções da biblioteca stdlib.h com malloc, que aloca espaço para um bloco de bytes consecutivos na memória RAM do computador e devolve o endereço desse bloco, calloc, o qual realiza a mesma ação do malloc, mas também faz com que o espaço alocado seja inicializado com zeros, realloc, que altera durante a execução do programa o tamanho de um bloco de bytes

alocado anteriormente, e free, que libera a memória ocupada por variáveis alocadas dinamicamente.

Os ponteiros são variáveis que armazenam o endereço de memória de outras variáveis, permitindo que sejam acessadas em diferentes partes do programa.

A modularização é a divisão do software em partes, chamadas módulos, que interagem entre si e cada um empenha uma funcionalidade específica - como um módulo com a declaração das funções e structs utilizadas, outro com o conteúdo das funções e outro com a execução delas. Isso permite melhorias na organização, manutenção e testabilidade do código. O Tipo Abstrato de Dado (TAD) é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados.

A lista linear duplamente encadeada é uma estrutura de dados composta por uma série de nós, os quais podem armazenar um ou mais dados, e possuem dois ponteiros, sendo um para o nó anterior e outro para o nó seguinte, permitindo que a lista seja percorrida a partir do seu início ou a partir do seu fim.

Os métodos de ordenação são algoritmos de manipulação de dados utilizados para organizá-los em uma ordem específica. Podem ser aplicadas em listas e vetores, por exemplo, e variam de acordo com complexidade e tempo de execução. Os métodos mais comuns são: bubble sort, selection sort, quick sort, insertion sort e merge sort.

## 3. Implementação de Algoritmos e Desenvolvimento

A organização das informações do estoque é crucial para um ótimo gerenciador. Nesse sentido, foi-se utilizado uma lista duplamente encadeada para armazenar as informações dos produtos presentes no estoque. Esse modo de guardar informações foi escolhido por trazer maior facilidade de em percorrer os elementos (tanto do elemento inicial para o final quanto do elemento final para o inicial).

Sabe-se também que o estoque de uma mercearia não é algo estático e está sujeito a constantes mudanças. Sempre existe a possibilidade de adicionar novas variedades de produtos, deixar de vender outras, fazendo como que seja necessário uma atualização do seu inventário. Com isso em mente, o gerenciador apresenta funções de adicionar e remover produto do estoque, para conseguir comportar essa dinamicidade do mercado.

Além disso, conferir a existência ou não de determinados produtos é de extrema importância para os comércios, a fim de não gerar erros de gestão, como oferecer a venda de algo que não existe, ou deixar de vender algo que existe e acabar perdendo o produto. Nesse pensamento, o sistema possui uma função de busca, para checar se realmente tem em estoque o produto procurado.

Organizar a disposição dos produtos é fundamental para atender as necessidades do cliente. Muitas vezes, o orçamento para compras é limitado e é importante que o cliente consiga encontrar facilmente os itens que estejam dentro de sua faixa de preço. Para ajudar nessa tarefa, o nosso software possui a funcionalidade de classificar os itens em ordem crescente de preço, tornando mais fácil para o cliente encontrar o que procura.

Somado a isso, o momento final de uma compra é o pagamento dela, e o cálculo correto deste valor total é crucial. Caso o valor seja calculado de forma errônea o proprietário da mercearia pode receber prejuízos financeiros (valor calculado ser menor que o correto), perder clientes, ou até sofrer processos judiciais (valor calculado acima do correto). De modo a prevenir isso, o gerenciador também apresenta uma função para calcular o valor total da compra, levando em consideração os itens e a quantidade de cada um a ser escolhida pelo comprador.

## 4. Resultados e Discussão

A Figura 1 ilustra o menu inicial, que mostra para o usuário as opções disponíveis para o gerenciamento do estoque.

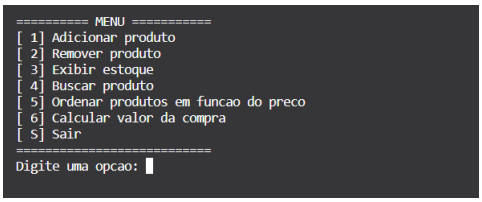


Figura 1. Menu inicial do software

A Figura 2 mostra como a função e adicionar um produto ao estoque é utilida pelo usuário final.

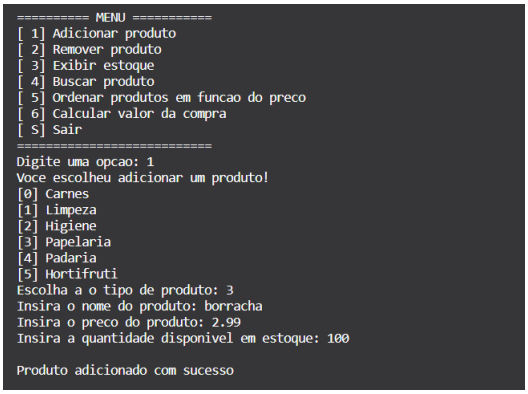


Figura 2. Adicionando um produto

A Figura 3 ilustra o que i gerenciaor exibe ao remover um item do estoque com sucesso.

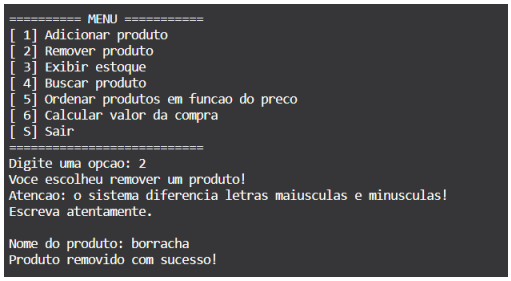


Figura 3. Remoção de produto no estoque

A Figura 4 ilustra o tratamento de erro que h´no gerenciador quando o usuário tenta remover um item que não está no estoque.

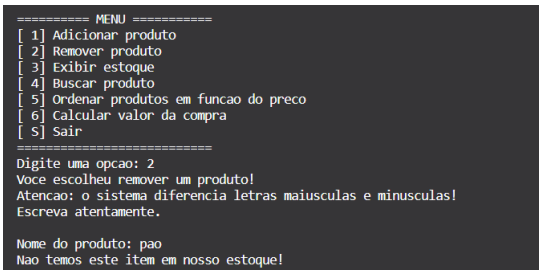


Figura 4. Tentataiva de remover produto inexistente

A Figura 5 ilustra o que o gerenciador mostra na tela para o usuário quando este busca bem sucedida, isto é, por um item que está no estoque.

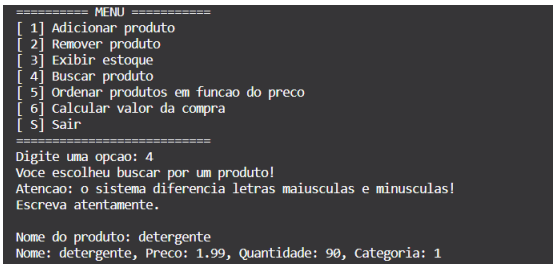


Figura 5. Busca realizada com sucesso

A Figura 6 ilustra o que o sistema mostra na tela para o usuário quando este busca por um item que não esta presente no estoque.

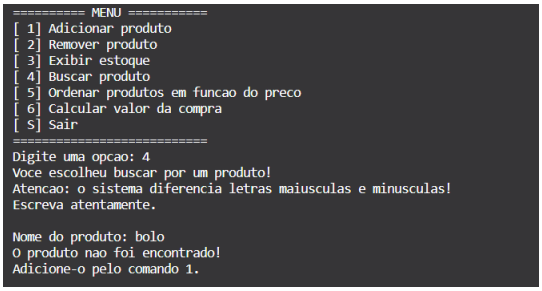


Figura 6. Erro ao não encontrar um produto no estoque

A Figura 7 ilustra como é mostrado na tela quando o usuário escolhe a opçõ de ordenar o estoque por ordem crescente de preço.

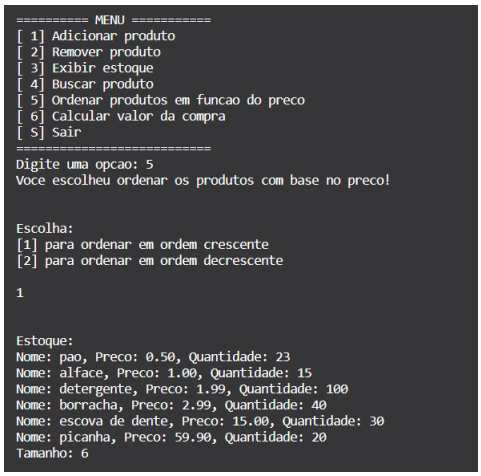


Figura 7. Ordenação crescente por preços

A Figura 8 ilustra como é mostrado na tela quando o usuário escolhe a opçõ de ordenar o estoque por ordem decrescente de preço.

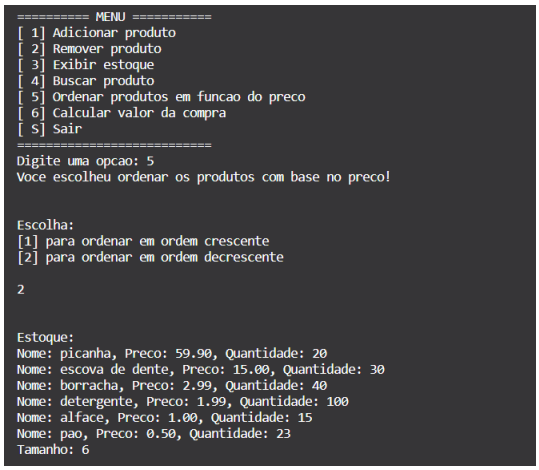


Figura 8. Ordenação decrescente por preços

A Figura 9 ilustra o tratamento de erro quando o usuário escolhe a opção e ordenr por preço, mas o estoque está vazio.

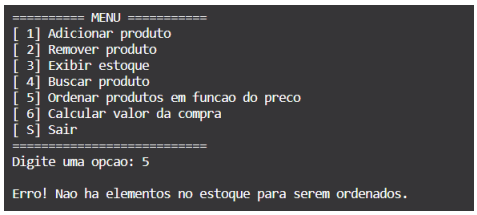


Figura 9. Ordenação falha pois não há estoque

A Figura 10 mostra um cálculo bem sucedido da compra de um usuário.

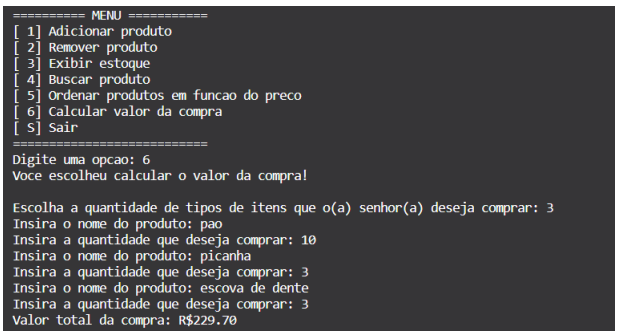


Figura 10. Sucesso ao comprar produtos

A Figura 11 ilustra o tratamento de erro quando um usuário escolhe a opção de fazer uma compra com o estoque vazio.

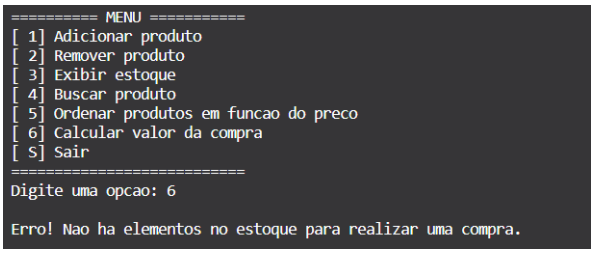


Figura 11. Tentativa falha de compra sem produtos previamente cadastrados

A Figura 12 mostra o resultado final da exibição do estoque cadastrado.

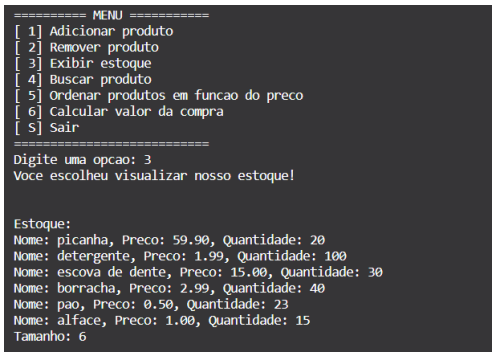


Figura 12. Sucesso ao exibir estoque

A Figura 13 mostra o tratamento de erro ao tentar exibir um estoque vazio.

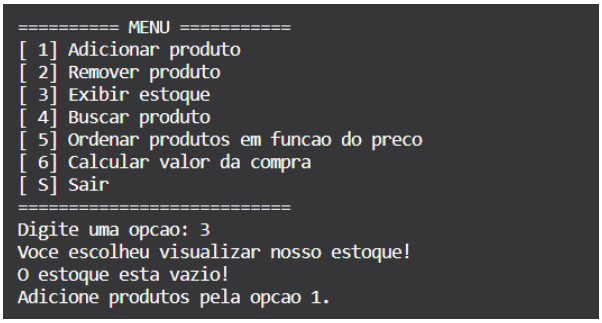


Figura 13. Tentativa falha de visualizar estoque vazio

A Figura 14 ilustra a saída do usuário do gerenciador de estoque.

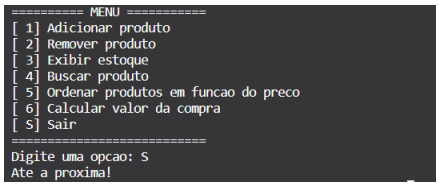


Figura 14. Saída do sistema

## 5. Conclusões

O resultado esperado foi obtido com êxito. Com o gerenciador foi possível categorizar, ordenar e controlar todos os produtos de maneira correta. Além disso, o trabalho foi de grande importância para aplicar os conhecimentos adquiridos ao longo das aulas de Algoritmo e Estrutura de Dados 1. Toda a equipe participou ativamente em todos os processos, desde a criação dads ideias, planejamento de funções, codificação, escrita até a apresentação do projeto.

## Referências

[Tenenbaum et *al*. (2004)] Tenenbaum, Aaron M., Yedidyah Langsam, and Moshe J. Augenstein. Estruturas de dados usando C. *Pearson Makron Books*, 2004.

# Apêndice A.
# Lista de códigos

O código foi separado em 3 arquivos diferentes. O primeiro, ”estoque.h”, é o header do nosso programa, isso significa que é onde há a inclusão de bibliotecas, definição dos Tipos Abstratos de Dados e declração das funções. O segundo, ”estoque.c”, é arquivo que todas as funções estão definidas. Por fim, o terceiro, ”main.c”, é a função principal do programa.

A Figura 15 mostra a primeira parte do header, na qual as bibliotecas são incluídas e os Tipos Abstratos de Dados (neste caso, lista e nó) utilizados são definidos.

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

//declaracoes de estruturas usadas para montar o estoque
typedef struct _no {
   double preco;
   int qtd, cat;
   char * nome;
   struct _no * prox;
   struct _no * ant;
} No;

typedef struct _lista {
   No * inicio, * fim;
   int tamanho;
} Lista;
```

Figura 15. Header 1

A Figura 16 mostra as declrações de todas as funções presentes no programa.

```c
// declaração de funções
void menu(void);
Lista * criar_lista_vazia(void);
No * criar_no(double p, int q, int c, char *nome);
void add_no_final(Lista * lista, No * no);
void remover_elemento(Lista * lista, char * nome);
No * buscar(Lista *lista);
void ordenacao_preco(Lista * lista);
void ordena_preco_cres(Lista * lista);
void ordena_preco_dec(Lista * lista);
void add_produto(Lista *lista_geral, Lista * vetor[]);
void imprimir (Lista *lista);
void remover_produto(Lista * lista_geral, Lista * vetor[]);
void valor_da_compra(Lista * lista);
```

Figura 16. Header 2

A Figura 17 mostra toda a função principal (main) do sistema.

```c
#include "estoque.h"

int main(void) {

   char ans;
   No * busca = NULL;
   Lista * lista_geral = criar_lista_vazia();

   Lista * carnes = criar_lista_vazia();
   Lista * limpeza = criar_lista_vazia();
   Lista * higiene = criar_lista_vazia();
   Lista * papelaria = criar_lista_vazia();
   Lista * padaria = criar_lista_vazia();
   Lista * hortifruti = criar_lista_vazia();
   Lista * vetor[6] = {carnes, limpeza, higiene, papelaria, padaria, hortifruti};
   //                  0       1        2        3         4        5

   do {
      menu(); //exibicao do rol de opcoes a serem selecionadas
      scanf("%c%*c", &ans);

      switch (ans) {
         case '1':
            printf("Voce escolheu adicionar um produto!\n");
            add_produto(lista_geral, vetor);
            break;

         case '2':
            printf("Voce escolheu remover um produto!\n");
            remover_produto(lista_geral, vetor);
            break;

         case '3':
            printf("Voce escolheu visualizar nosso estoque!\n");
            imprimir(lista_geral);
            break;

         case '4':
            printf("Voce escolheu buscar por um produto!\n");
            busca = buscar(lista_geral);
            if(busca!=NULL) printf("Nome: %s, Preco: %.2lf, Quantidade: %d, Categoria: %d",
busca->nome, busca->preco, busca->qtd, busca->cat);
            else printf("O produto nao foi encontrado!\nAdicione-o pelo comando 1.\n");
            break;

         case '5':
            if(lista_geral->tamanho==0){
                printf("\nErro! Nao ha elementos no estoque para serem ordenados.\n");
                break;
            }
            printf("Voce escolheu ordenar os produtos com base no preco!\n");
            ordenacao_preco(lista_geral);
            break;

         case '6':
            if(lista_geral->tamanho==0){
                printf("\nErro! Nao ha elementos no estoque para realizar uma compra.\n");
                break;
            }
            printf("Voce escolheu calcular o valor da compra!\n");
            valor_da_compra(lista_geral);
            break;

         case 'S':
            printf("Ate a proxima!\n");
            break;

         default:
            printf("Opcao invalida!\n");
            break;
      }

   } while (ans != 'S'); //'S' e o comando de saida

   return 0;
}
```

Figura 17. Main

A Figura 18 mostra o funcionamento da função de exibir o menu.

```c
#include "estoque.h"

void menu(void) { // funcao que exibe um rol de opcoes para o usuario
   printf("\n========== MENU ==========\n");
   printf("[ 1] Adicionar produto\n");
   printf("[ 2] Remover produto\n");
   printf("[ 3] Exibir estoque\n");
   printf("[ 4] Buscar produto\n");
   printf("[ 5] Ordenar produtos em funcao do preco\n");
   printf("[ 6] Calcular valor da compra\n");
   printf("[ S] Sair\n");
   printf("==========================\n");
   printf("Digite uma opcao: ");
}
```

Figura 18. Menu

A Figura 19 mostra o funcionamento da função de criar

uma lista vazia.

```c
Lista * criar_lista_vazia(void) { // funcao que cria uma lista, vazia, que ira estocar produtos
    Lista * lista = NULL;
    lista = (Lista *) malloc(sizeof(Lista));

    if (lista == NULL) return NULL;

    lista->inicio = NULL;
    lista->fim = NULL;
    lista->tamanho = 0;
    return lista;
}
```

Figura 19. Função Criar Lista Vazia

A Figura 20 mostra o funcionamento da função de criar um nó - neste sisetma, um produto é trtdo como um nó de uma lista duplamente encadeada.

```c
No * criar_no(double p, int q, int c, char *nome){ //funcao que cria os nos relativos as listas de produtos
    No * no = NULL;
    no = (No *) malloc(sizeof(No));

    if (no == NULL) return NULL;

    no->preco = p;
    no->qtd = q;
    no->cat = c;

    no->nome = NULL;
    no->nome = (char *) malloc((strlen(nome)+1)*sizeof(char));
    strcpy(no->nome, nome);

    no->prox = NULL;
    no->ant = NULL;

    return no;

}
```

Figura 20. Função Criar Nó

A Figura 21 mostra o funcionamento da função de adicionar um produto ao final da lista.

```c
void add_no_final(Lista * lista, No * no) { //funcao que adiciona, ao final da lista, um produto recem-adicionado
    if (lista == NULL) return;

    if (lista->inicio == NULL) { // inicio da lista
        lista->inicio = no;
        lista->fim = no;

    } else {
        no->ant = lista->fim;
        lista->fim->prox = no;
        lista->fim = no;
    }

    lista->tamanho++;
}
```

Figura 21. Função Adicionar no Final

A Figura 22 mostra o funcionamento da função de adicionar um produto tanto à lista de todos os itens qunto da list e sua respectiva catetgoria.

```c
void add_produto(Lista * lista_geral, Lista * vetor[]){ //cria o produto requerido, e ao final chama a funcao que adiciona o item ao final
    int cat, qtd;
    double preco;
    char nome[100];
    No * produto = NULL;

    printf("[0] Carnes\n[1] Limpeza\n[2] Higiene\n[3] Papelaria\n[4] Padaria\n[5] Hortifruti\n");
    printf("Escolha a o tipo de produto: ");
    scanf("%d%*c", &cat);

    printf("Insira o nome do produto: ");
    scanf("%[^\n]%*c", nome);

    printf("Insira o preco do produto: ");
    scanf("%lf%*c", &preco);

    printf("Insira a quantidade disponivel em estoque: ");
    scanf("%d%*c", &qtd);

    produto = criar_no(preco, qtd, cat, nome);
    add_no_final(vetor[cat], produto);
    add_no_final(lista_geral, produto);

    printf("\nProduto adicionado com sucesso\n");
}
```

Figura 22. Função Adicionar Produto

A Figura 23 mostra o funcionamento da função de remover um elemento de uma lista, baseado em seu nome.

```c
void remover_elemento(Lista * lista, char * nome) { //funcao que remove da lista o produto escolhido
    No *remover = NULL;
    No *aux;

    if(lista->inicio != NULL){
        if(strcmp(lista->inicio->nome, nome) == 0){
            remover = lista->inicio;
            lista->inicio = remover->prox;
            if(lista->inicio != NULL)
                lista->inicio->ant = NULL;

        } else {
            aux = lista->inicio;
            while(aux->prox != NULL && strcmp(aux->prox->nome, nome) != 0)
                aux = aux->prox;

            if(aux->prox != NULL){
                remover = aux->prox;
                aux->prox = remover->prox;
                if(aux->prox != NULL)
                    aux->prox->ant = aux;

            }
        }
    }
}
```

Figura 23. Função Remover Elemento

A Figura 24 mostra o funcionamento da função de remover um prouto, tanto da lista de todos os produtos quanto da lista de sua respectiva categoria.

```c
void remover_produto(Lista * lista_geral, Lista * vetor[]) { //funcao que "chama" a funcao remover_elemento para lista geral e especifica
    No * remover = NULL;
    remover = buscar(lista_geral);

   if(lista_geral->inicio == NULL) {
      printf("Nosso estoque esta vazio!\n");
      printf("Adicione itens pela opcao 1.\n");
      return;
    }

    if(remover == NULL) {
      printf("Nao temos este item em nosso estoque!");
      return;
    }

    remover_elemento(lista_geral, remover->nome);
    remover_elemento(vetor[remover->cat], remover->nome);
    lista_geral->tamanho--;
    vetor[remover->cat]->tamanho--;
    free(remover);

    printf("Produto removido com sucesso!\n");
}
```

Figura 24. Função Remover Produto

A Figura 25 mostra o funcionamento da função de busca por um produto.

```c
No *buscar(Lista *lista){ // funcao que realiza a busca por um produto dentro do inventario cadastrado
    No * no = NULL;
    No * aux;
    char nome[100];
    printf("Atencao: o sistema diferencia letras maiusculas e minusculas!\nEscreva atentamente.\n\n"); //para prevenir possiveis erros
    printf("Nome do produto: ");
    scanf("%[^\n]%*c", nome);

    aux = lista->inicio;
    while(aux && strcmp(aux->nome, nome) != 0)
        aux = aux->prox;

    if(aux)
        no = aux;
    return(no);
}
```

Figura 25. Função Buscar

A Figura 26 mostra o funcionamento da função de ordenar a lista com todo o estoque.

```c
void ordenacao_preco(Lista * lista){
    int op;

    printf("\n\nEscolha:\n[1] para ordenar em ordem crescente\n[2] para ordenar em ordem decrescente\n\n");
    scanf("%d%*c", &op);

    switch (op){
    case 1:
        ordena_preco_cres(lista);
        imprimir(lista);
        break;

    case 2:
        ordena_preco_dec(lista);
        imprimir(lista);
        break;

    default:
        printf("\nErro! Opcao invalida.\n");
        break;
    }


}
```

Figura 26. Função Ordenação

A Figura 27 mostra o funcionamento da função de ordenar a lista com todo o estoque em ordem crescente.

```c
void ordena_preco_cres(Lista * lista) { //ordena, por selection sort, o estoque em ordem crescente de preco
    No *i, *j;
    double tempPreco;
    int tempQtd, tempCat;
    char *tempNome;

    for(i = lista->inicio; i != NULL; i = i->prox) {
        for(j = i->prox; j != NULL; j = j->prox) {
            if(j->preco < i->preco) {
                tempPreco = i->preco;
                i->preco = j->preco;
                j->preco = tempPreco;
                tempQtd = i->qtd;
                i->qtd = j->qtd;
                j->qtd = tempQtd;
                tempCat = i->cat;
                i->cat = j->cat;
                j->cat = tempCat;
                tempNome = i->nome;
                i->nome = j->nome;
                j->nome = tempNome;
            }
        }
    }
}
```

Figura 27. Função Ordenação Crescente

A Figura 28 mostra o funcionamento da função de ordenar a lista com todo o estoque em ordem decrescente.

```c
void ordena_preco_dec(Lista * lista){ //assim como a funcao anterior, o rdena por preco, mas por decrescencia de preco
 No *i, *j;
    double tempPreco;
    int tempQtd, tempCat;
    char *tempNome;

    for(i = lista->inicio; i != NULL; i = i->prox) {
        for(j = i->prox; j != NULL; j = j->prox) {
            if(j->preco > i->preco) {
                tempPreco = i->preco;
                i->preco = j->preco;
                j->preco = tempPreco;
                tempQtd = i->qtd;
                i->qtd = j->qtd;
                j->qtd = tempQtd;
                tempCat = i->cat;
                i->cat = j->cat;
                j->cat = tempCat;
                tempNome = i->nome;
                i->nome = j->nome;
                j->nome = tempNome;
            }
        }
    }

}
```

Figura 28. Função Ordenação Decrescente

A Figura 29 mostra o funcionamento da função de ordenar a lista com todo o estoque em ordem crescente.

```c
void valor_da_compra(Lista * lista){ //calcula o valor da compra do cliente
    int qtd_geral, qtd_indv, i;
    double valor = 0;
    char nome[100];
    No * produto = NULL;
    No * aux;

    printf("\nEscolha a quantidade de tipos de itens que o(a) senhor(a) deseja comprar: ");
    scanf("%d%*c", &qtd_geral);

    for(i = 0; i < qtd_geral; i++){
        printf("Insira o nome do produto: ");
        scanf("%[^\n]%*c", nome);
        printf("Insira a quantidade que deseja comprar: ");
        scanf("%d%*c", &qtd_indv);

        aux = lista->inicio;
        while(aux && strcmp(aux->nome, nome) != 0) //buscando o produto escolhido
            aux = aux->prox;

        if(aux) //verifica se o produto existe
            produto = aux;

        if(produto != NULL){
            if(0 < qtd_indv <= produto->qtd) //verifica se existe a quantidade escolhida no estoque
                valor += (produto->preco * qtd_indv);
            else {
                if(qtd_indv <= 0) printf("A quatidade deve ser maior que 0.\n");
                if(qtd_indv > produto->qtd) printf("Infelizmente nao temos essa quantidade em estoque.\n");
            }

        } else printf("Produto fora de estoque.\n");
    }

    printf("Valor total da compra: R$%.2lf", valor);
}
```

Figura 29. Função Valor da Compra

A Figura 30 mostra o funcionamento da função de mostrar a lista de produtos na tela.

```c
void imprimir (Lista *lista) { //exibe, na tela, a lista de produtos estocados
    No * aux = lista->inicio;
    if(lista->tamanho==0){
        printf("O estoque esta vazio!\nAdicione produtos pela opcao 1.\n");
        return;
    }
    printf("\n\nEstoque:\n");
    while(aux != NULL) {
        printf("Nome: %s, Preco: %.2lf, Quantidade: %d\n", aux->nome, aux->preco, aux->qtd);
        aux = aux->prox;
    }
    printf("Tamanho: %d\n", lista->tamanho);
}
```

Figura 30. Função Imprimir