INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS SÃO JOÃO EVANGELISTA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GABRIEL KÁICON BATISTA HILÁRIO

TRABALHO PRÁTICO II

SÃO JOÃO EVANGELISTA
OUTUBRO - 2022

GABRIEL KÁICON BATISTA HILÁRIO

SISTEMA DE DELIVERY

SÃO JOÃO EVANGELISTA

OUTUBRO - 2022

SUMÁRIO

1. INTRODUÇÃO

Este trabalho prático foi documentado para que seja avaliado em conjunto com os códigos na linguagem C/C++, exigido pelo docente Eduardo Augusto da Costa Trindade, dentro da disciplina de Algoritmos e Estruturas de Dados I, ministrada pelo mesmo. Porém a documentação tem cunho expositivo, onde é descrito as funcionalidades do programa, com testes, e desenvolvimento de novas linhas de raciocínio lógico para realização do trabalho prático.

1.1. Objetivo Geral

Este trabalho tem como objetivo geral apresentar na prática os conhecimentos adquiridos nas aulas de Algoritmos e Estruturas de Dados I, a respeito de Pilha, Listas com arranjo (Lista Sequencial), Tipos Abstratos de Dados (TADs) e Manipulação de Arquivos, utilizando a linguagem de C++ para escrita dos códigos.

1.2. Objetivos Específicos

Esse trabalho tem como objetivos específicos:

- Apresentar conhecimentos Pilha;
- Aprimorar conhecimentos em lista com arranjo;
- Desenvolver novas linhas de raciocínio para outros tipos de lógicas de aplicativos.

1.3. Justificativa

Ao iniciar os estudos de Algoritmos e Estruturas de Dados I, vemos os conteúdos de Ponteiros, TADs, Manipulação de Arquivos, Listas com Arranjo e com Ponteiros, e agora vimos Pilha.

Vemos em ponteiros, a manipulação de valores da variável por meio do endereço de memória, utilizando ponteiros. Vemos em arquivos, os comandos básicos de leitura e gravação de dados em um arquivo por meio de objetos da biblioteca *fstream*, o *ifstream* para leitura e o *ofstream* para gravação. Posteriormente vemos um conteúdo mais amplo de listas, que consiste na inserção de itens em uma lista. Seja ela qualquer um dos dois tipos apresentados, no caso a lista com arranjo, também chamada de lista sequencial que é com alocação estática, ou seja, possui um limite pré-definido, podendo ter um número limitado de itens, e a lista encadeada ou lista com ponteiro que é a com alocação dinâmica, ou seja, não possui um limite pré-definido podendo ter um número infinito de itens. Por fim vemos o conteúdo de Pilha com Arranjo, que funciona como uma pilha na vida real, onde empilhamos as coisas,

e para retirar o primeiro item empilhado, devemos remover todos os itens empilhados após ele, para remove-lo, e para remover o segundo, devemos remover todos os itens inseridos após ele, para então remover o segundo, e isso serve para todo e qualquer item empilhado. Resumindo, o primeiro que entra é o último que sai.

No trabalho, é exigido que usemos pilha, antes devemos cadastrar pedidos em uma pilha com arranjo, e utilizei uma lista com arranjo com produtos cadastrados de uma lista. Nem todos os conteúdos foram aplicados para realização do trabalho, apenas os conteúdos de pilha, lista com arranjo e arquivos, criando um minissistema de delivery que nos permite empilhar pedidos contendo produtos.

Tendo isso tudo em vista, o trabalho foi exigido para que seja possível desenvolver o raciocínio lógico quanto a aplicação dessa estrutura, e com o bônus de novamente aplicar elas em conjunto.

2. DESENVOLVIMENTO

Nesta seção do documento é apresentado, os conceitos aprendidos e o desenvolvimento do trabalho em si, na linguagem C++.

2.1. Conceitos Aplicados

Explicação sucinta dos conceitos de Pilha, Lista sequencial, arquivos e Tipos Abstratos de Dados (TADs).

2.1.1. Tipos Abstratos de Dados

As estruturas utilizadas como registro para criação de objetos, e as funções de manipulação dessas estruturas, ambas compõem uma TAD. Declaramos esses modelos como *structs*, elas são representações de qualquer coisa no mundo real, sendo ela lógica, abstrata ou física, como por exemplo uma pessoa, que é algo físico, ou um filme digital, que é algo lógico/abstrato, veja o exemplo na figura 1. Cada um tem suas características específicas, uma pessoa nome, sexo, idade, CPF, altura, dentre outras, e um filme título, linguagem, elenco, personagens, duração, categoria, ano de lançamento, dentre outros, e tudo isso pode ser definida dentro de uma struct para cada um deles. Resumindo uma Struct é uma espécie de variável modelo para cadastrar diferentes itens, dentro de um software escrito em C/C++. Acompanhado das structs temos as funções para manipulação dos dados dessa lista, e desses itens, que será visto no próximo tópico.

Figura 1 – Struct exemplo, sem ligação com o trabalho

```
6   typedef struct Horario{
7       int hora;
8       int min;
9   };
10
11  typedef struct Data{
12      int dia;
13      int mes;
14      int ano;
15  };
16
17  typedef struct Compromisso{
18      char descricao[100];
19      Data dt;
20      Horario hr;
21  };
```

2.1.2. Pilha

Figura 2 – Pilha

Na figura 2, vemos um desenho esquemático de como é a estrutura de dados da pilha. O limite dela é definida na hora da criação, o N seria o topo – 1, seria o último item empilhado. Na figura 3 vemos o processo de empilhamento de itens em uma pilha, e na figura 4 vemos o desempilhamento.

Figura 3 – Empilhamento de itens

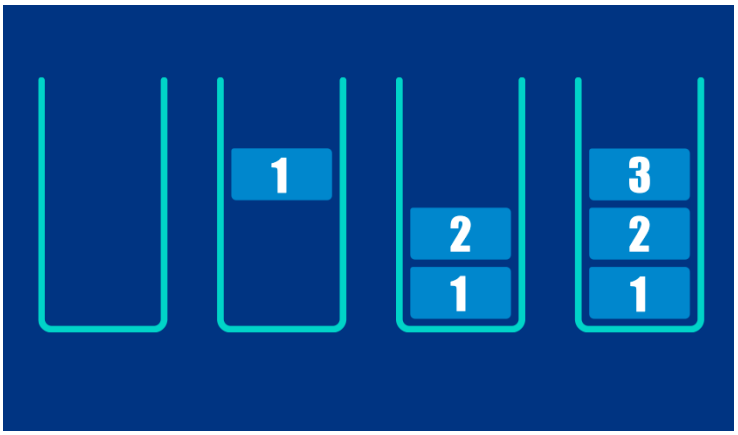
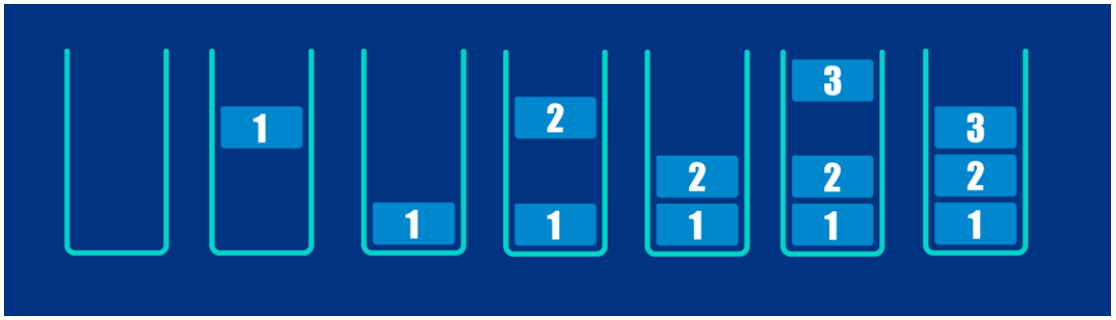
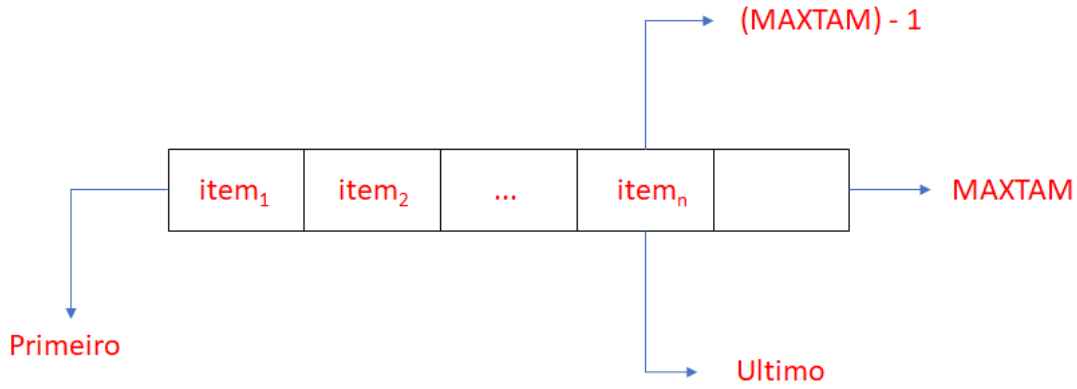


Figura 4 – Desempilhamento de itens



2.1.3. Lista Sequencial

Figura 4 – Lista Sequencial



Na figura 4, vemos um desenho esquemático de como seria uma lista sequencial, e o limite dela, é o MAXTAM, uma variável didática muito comum de ser usada, e sua função é delimitar o tamanho da lista. Ela é um vetor, porém a tipagem de dados a ser inserida podem ser as *structs*, podendo armazenar mais de um tipo de variável dentro de uma posição. Os elementos são inseridos dentro do índice do vetor no ultimo, que seria o apontador para última posição até o momento, e assim sucessivamente, até que a Lista alcance o tamanho máximo, que seria o valor definido para o MAXTAM.

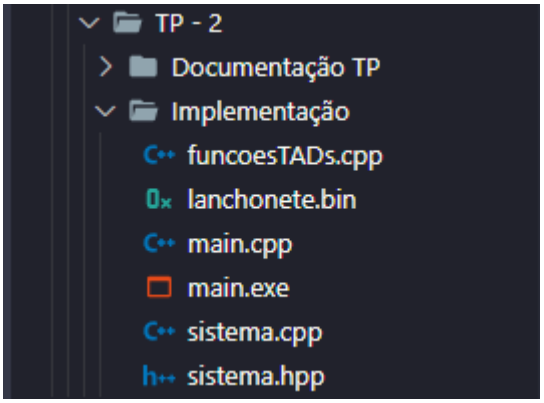
2.1.4. Arquivos

A parte de manipulação de arquivos é mais abstrata, não é convencional para ilustrar ela, mas podemos usar uma analogia para compreender, imagine-se estudando para uma prova, onde se usa um livro e uma folha de papel em branco, o papel é seu arquivo, e o livro é seu programa, quando você quer guardar algo importante do livro, você lê o que está nele e escreve na folha, isso seria semelhante às funções do *ofstream*, que pega os dados digitados no programa e escreve no arquivo, independente da extensão deste. Imagine novamente, com um caderno e uma folha apenas, o caderno é o programa, e a folha é o arquivo com o que você leu dele, e não estava em branco quando começou tendo algo escrito na folha, você realiza a leitura e exibe isso no caderno, seria semelhante às funções do *ifstream*, que pega os dados do arquivo e exibe na tela.

2.2. Implementação

Utilizei uma modularização que está na figura 5, dividindo em 6 arquivos, 3 arquivos *.cpp, 1 arquivo *.bin, 1 arquivo *.hpp, e 1 arquivo *.exe.

Figura 5



No arquivo funcoesTADs.cpp, tenho todas as funções das TADs de pilha (Apêndice A) e de lista (Apêndice B). No sistema.hpp (Apêndice C), contém as TADs de pilha e lista, além do cabeçalho das funções das TADs, do arquivo.cpp e da manipulação de arquivos. A manipulação de arquivos (Apêndice D) utilizadas foi mesma de anteriormente, porém adaptado para esse trabalho prático.

Das funções apresentadas nas TADS, temos 2 diferentes do padrão, a de ordenar pilha, na figura 6, e uma que recolhe o pedido que está topo da pilha, na figura 7.

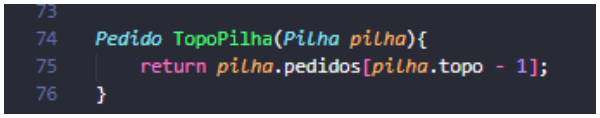
A função OrdenaPilha, serve para ordenar uma pilha com base na distancia dos pedidos contidos nessa pilha.

Figura 6

```
56  Pilha OrdenaPilha(Pilha pilha){ //vou chamar essa pilha do paramentro de parametroPilha nos comentários
57      Pilha auxPilha;
58      InicializaPilha(&auxPilha);
59
60      while (!PilhaVazia(pilha)){
61          Pedido pedido = TopoPilha(pilha); //pega o topo da parametroPilha e armazena na variavel pedido
62          Desempilha(&pilha); //desempilha da parametroPilha
63
64          //enquanto o topo da auxPilha tiver a distancia menor que a distancia do pedido da parametroPilha
65          while (!PilhaVazia(auxPilha) && TopoPilha(auxPilha).distancia < pedido.distancia){
66              Empilha(&pilha, TopoPilha(auxPilha)); //empilha na pilhaparametro o que tá no topo da pilha auxiliar
67              Desempilha(&auxPilha);//removo o ultimo item da pilha auxiliar
68          }//não vai sair do laço enquanto, o pedido for maior que o topo da auxiliar
69          Empilha(&auxPilha, pedido); //quando sair significa que achou um valor maior, aí no meio do laço os itens menores são empilhados novamente
70      }
71      return auxPilha;
72  }
```

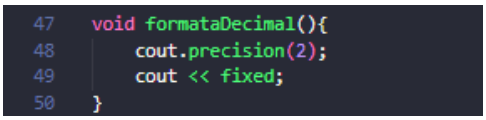
A função TopoPilha, serve para guardar o último pedido inserido na pilha.

Figura 7



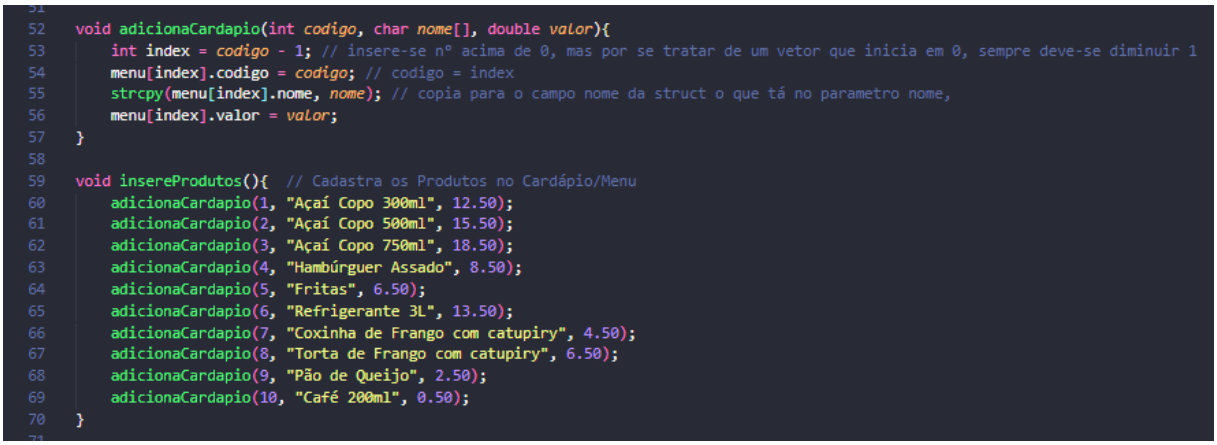
A função formataDecimal na figura 8, serve para deixar os valores de tipo double, em 2 casas decimais.

Figura 8



Na figura 9, vemos duas funções agindo em conjunto. A função adicionaCardapio, serve para que seja possível adicionar produtos no vetor de Produtos chamado menu, que está no arquivo sistema.hpp (Apêndice C), e a função insereProdutos, faz uso da função anterior para inserir os produtos de fato, no cardápio no momento da inicialização.

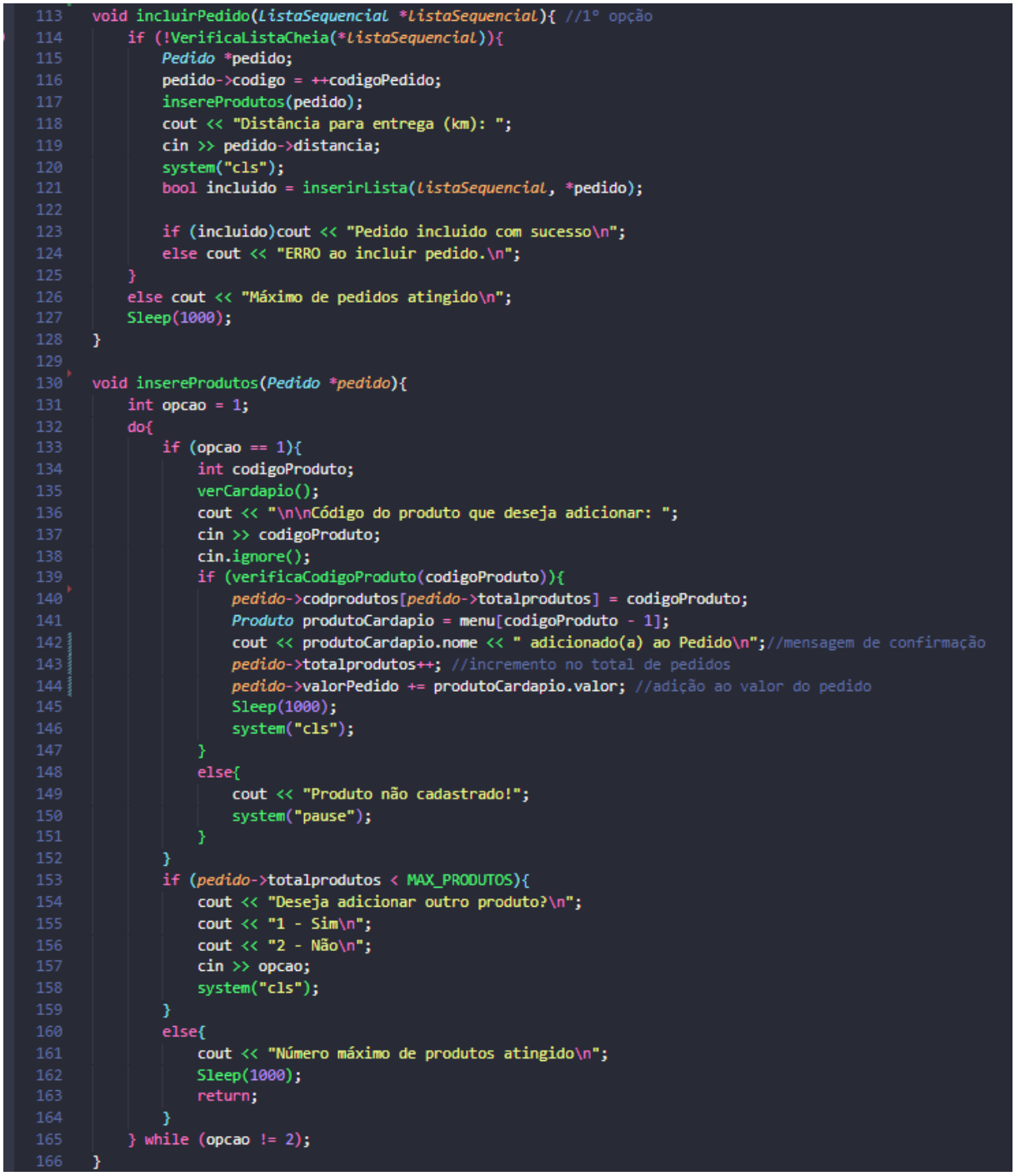
Figura 9



A Menu (Apêndice F) contém as opções exigidas no trabalho. A começar pela primeira opção, a Inclusão de pedido mostrada na figura 10. O código começa verificando se a lista está cheia, se tiver, a inserção não será feita, é criada uma variável pedido, que recebe um código de acordo com a variável que evita a repetição de códigos, para que o usuário não cometa o erro de digitar um código já existente, logo em seguida vai para função insereProdutos, que tem o pedido como parâmetro, para que sejam inseridos produtos apenas naquele pedido, lá dentro da função, é inserido um código já existente, e é feita uma verificação de existência desse código, feita pela função verificaCodigoProduto (Apêndice G), se o código existe, o produto é

incluído no pedido, e depois disso, vem a mensagem de confirmação, e se quer que insere um novo produto, é claro, respeitando o limite de produtos por entrega.

Figura 10



```
113  void incluirPedido(ListaSequencial *listaSequencial){ //1° opção
114      if (!VerificaListaCheia(*listaSequencial)){
115          Pedido *pedido;
116          pedido->codigo = ++codigoPedido;
117          insereProdutos(pedido);
118          cout << "Distância para entrega (km): ";
119          cin >> pedido->distancia;
120          system("cls");
121          bool incluido = inserirLista(listaSequencial, *pedido);
122
123          if (incluido)cout << "Pedido incluido com sucesso\n";
124          else cout << "ERRO ao incluir pedido.\n";
125      }
126      else cout << "Máximo de pedidos atingido\n";
127      Sleep(1000);
128  }
129
130  void insereProdutos(Pedido *pedido){
131      int opcao = 1;
132      do{
133          if (opcao == 1){
134              int codigoProduto;
135              verCardapio();
136              cout << "\n\nCódigo do produto que deseja adicionar: ";
137              cin >> codigoProduto;
138              cin.ignore();
139              if (verificaCodigoProduto(codigoProduto)){
140                  pedido->codprodutos[pedido->totalprodutos] = codigoProduto;
141                  Produto produtoCardapio = menu[codigoProduto - 1];
142                  cout << produtoCardapio.nome << " adicionado(a) ao Pedido\n";//mensagem de confirmação
143                  pedido->totalprodutos++; //incremento no total de pedidos
144                  pedido->valorPedido += produtoCardapio.valor; //adição ao valor do pedido
145                  Sleep(1000);
146                  system("cls");
147              }
148              else{
149                  cout << "Produto não cadastrado!";
150                  system("pause");
151              }
152          }
153          if (pedido->totalprodutos < MAX_PRODUTOS){
154              cout << "Deseja adicionar outro produto?\n";
155              cout << "1 - Sim\n";
156              cout << "2 - Não\n";
157              cin >> opcao;
158              system("cls");
159          }
160          else{
161              cout << "Número máximo de produtos atingido\n";
162              Sleep(1000);
163              return;
164          }
165      } while (opcao != 2);
166  }
```

A segunda opção é a de Listar pedidos, onde a função listapedidos, na figura 11 é chamada, que simplesmente verifica se lista está vazia, e se não tiver, ele exibe a lista existente.

Figura 11

```
168  void listarPedidos(ListaSequencial listaSequencial){ // 2° opção
169      if (!VerificaListaVazia(listaSequencial)){
170          imprimeLista(listaSequencial);
171          system("pause");
172      }
173      else cout << "Nenhum pedido cadastrado\n";
174      Sleep(1000);
175  }
```

A terceira opção é a de ver o cardápio, onde a função verCardapio, na figura 12 é chamada, que imprime os produtos cadastrados no inicio da execução do programa.

Figura 12

```
177  void verCardapio(){ // 3° opção
178      cout << "Menu:\n";
179      for (int i = 0; i < MAX_PRODUTOS_CARDAPIO; i++){
180          Produto produto = menu[i];
181          cout << "\tCódigo: " << produto.codigo << endl;
182          cout << "\tNome: " << produto.nome << endl;
183          cout << "\tValor do produto: R$" << produto.valor << endl;
184          cout << '\n';
185      }
186      system("pause");
187  }
```

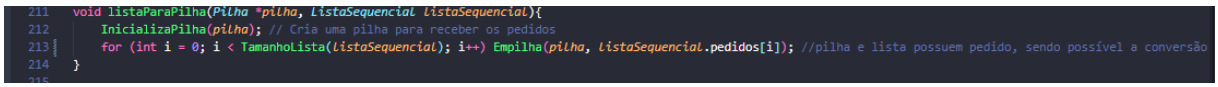
A quarta opção é a de consultar pedido, onde a função consultaPedido, na figura 13, procura por um pedido dentro da lista, e imprime ele, caso ele seja encontrado.

Figura 13

```
189  void consultarPedido(ListaSequencial listaSequencial){
190      if (!VerificaListaVazia(listaSequencial)){
191          int codigo;
192          cout << "Consulta pedido\n";
193          cout << "Código do pedido: ";
194          cin >> codigo;
195          cin.ignore();
196          system("cls");
197          for (int i = 0; i < TamanhoLista(listaSequencial); i++){
198              if (listaSequencial.pedidos[i].codigo == codigo){
199                  imprimePedido(listaSequencial.pedidos[i]);
200                  system("pause");
201                  return;
202              }
203          }
204          cout << "Pedido não encontrado\n";
205          Sleep(1000);
206      }
207      else cout << "Nenhum pedido cadastrado\n";
208          Sleep(1000);
209  }
```

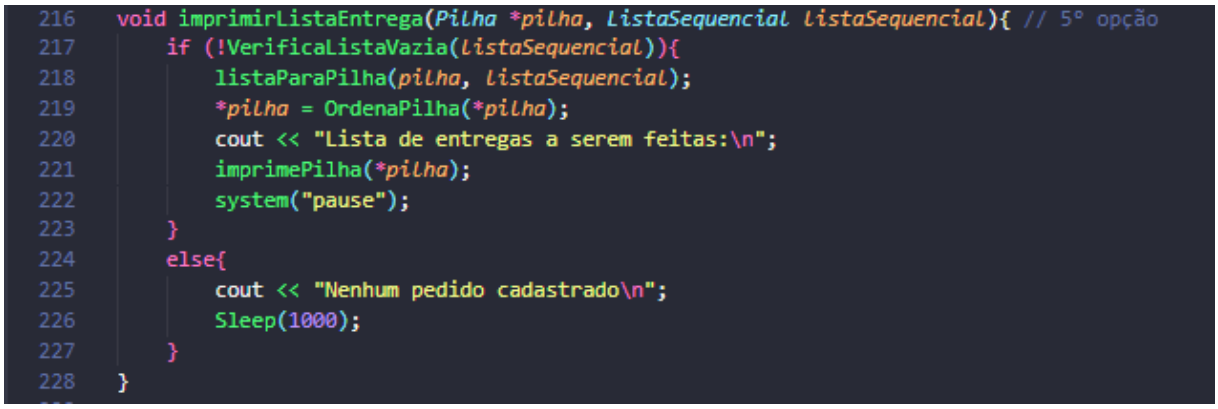
Temos a quinta opção, que é a de imprimir a lista de entrega, para isso temos função que converte uma lista para uma pilha na figura 14, para que seja possível imprimir a lista de entregas (que na verdade é uma pilha de entrega) na figura 15. A conversão de uma estrutura de dados em outra é possível nesse caso, pois ambas as estruturas possuem pedidos. Para mostramos a pilha de entregas, devemos ordenar ela, da distancia mais curta primeiro, e a mais longa por último.

Figura 14



```
void listaParaPilha(Pilha *pilha, ListaSequencial listaSequencial){
    InicializaPilha(pilha); // Cria uma pilha para receber os pedidos
    for (int i = 0; i < TamanhoLista(listaSequencial); i++) Empilha(pilha, listaSequencial.pedidos[i]); //pilha e lista possuem pedido, sendo possível a conversão
}
```

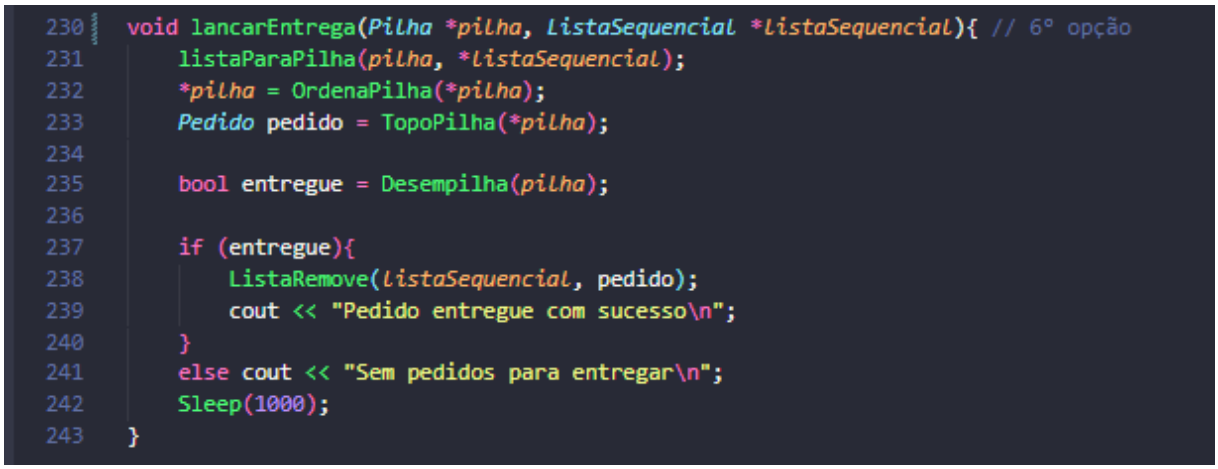
Figura 15



```
void imprimirListaEntrega(Pilha *pilha, ListaSequencial listaSequencial){ // 5° opção
    if (!VerificaListaVazia(listaSequencial)){
        listaParaPilha(pilha, listaSequencial);
        *pilha = OrdenaPilha(*pilha);
        cout << "Lista de entregas a serem feitas:\n";
        imprimePilha(*pilha);
        system("pause");
    }
    else{
        cout << "Nenhum pedido cadastrado\n";
        Sleep(1000);
    }
}
```

Temos a sexta opção, que é a função de lançar entrega vista na figura 16, que de forma resumida, ela simplesmente remove o topo da pilha de entregas.

Figura 16



```
void lancarEntrega(Pilha *pilha, ListaSequencial *listaSequencial){ // 6° opção
    listaParaPilha(pilha, *listaSequencial);
    *pilha = OrdenaPilha(*pilha);
    Pedido pedido = TopoPilha(*pilha);

    bool entregue = Desempilha(pilha);

    if (entregue){
        ListaRemove(listaSequencial, pedido);
        cout << "Pedido entregue com sucesso\n";
    }
    else cout << "Sem pedidos para entregar\n";
    Sleep(1000);
}
```

3. CONCLUSÃO

Ao longo do trabalho tive várias ideias de como resolver o problema, essa parte de pilha é mais interessante, reutilizei os códigos padrões das TADs e fiz as devidas alterações, reutilizei um pouco do código do último trabalho que fiz, para a manipulação de arquivos, e reutilizei textos do meu 1° trabalho prático. Fiz uso dos slides disponibilizados pelo professor para sanar dúvidas a respeito de pilha.

Coloquei o trabalho em modularização separando-o em 6 arquivos:

- Dois arquivos *.cpp:
  - sistema.cpp, com as funções exigidas
  - funcoesTADs, com as funções das TADs
  - main.cpp com a execução das funções,
- Um arquivo *.hpp, o sistema.hpp, onde tinha as structs e o cabeçalho das funções,
- Um arquivo *.bin, lanchonete.bin onde os funcionários eram salvos,
- Um arquivo *.exe, a main.exe, que seria o executável do código, onde as funções compilavam.

Creio eu que meu desenvolvimento foi bom nesse trabalho, desenvolvi novas formas de raciocínio e tive que abstrair e diminuir muitas coisas para que o trabalho ficasse leve, sem ser muito extenso. Consegui atingir meus objetivos, e explicar bem o que cada função faz, apresentar conhecimentos em Pilha, e com confiança aprimorar meus conhecimentos em lista sequencial e fazer uso da mesma dentro do meu programa novamente, e aplicar isso em um software de *delivery*, que aumenta minha percepção para conseguir criar um software de entregas. Estou feliz e satisfeito com o resultado, e ele atendeu às minhas expectativas além do que eu esperava.

4. REFERÊNCIAS

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Arquivos. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/146487/mod_resource/content/1/Aula%203%20-%20Arquivos.pdf . Acesso em: 7 de outubro de 2022.

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados - Listas. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/146499/mod_resource/content/1/Aula%206%20-%20Listas.pdf . Acesso em: 7 de outubro de 2022.

TRINDADE. Eduardo. Algoritmos e Estrutura de Dados – Pilha. 2022. Apresentação PDF. Disponível em: https://ead.ifmg.edu.br/saojoaoevangelista/pluginfile.php/146500/mod_resource/content/1/Aula%207%20-%20Listas%20Encadeadas.pdf . Acesso em: 7 de outubro de 2022.

Ana Paula Andrade. O que é e como funciona a Estrutura de Dados Pilha. **TREINAWEB.** Outubro de 2020. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-e-como-funciona-a-estrutura-de-dados-pilha. Acesso em: 7 de outubro de 2022.

Link do código no *Github*: https://github.com/gKaicon/Arquivos_C_and_C-withClasses/tree/main/AEDs%20I/TP/TP%20-%202

5. APÊNDICES

5.1. APÊNDICE A – TADs de Pilha

AEDs I > TP > TP - 2 > Implementação > funcoesTADs.cpp > OrdenaPilha(Pilha)

You, há 2 minutos | 1 author (You)

```cpp
#include "sistema.hpp"
//----------------------------------------TADs Pilha----------------------------------------
void InicializaPilha(Pilha *pilha){
    pilha->topo = 0;
}

bool PilhaVazia(Pilha pilha){
    return pilha.topo == 0;
}

bool PilhaCheia(Pilha pilha){
    return pilha.topo == MAX_ENTREGA;
}

void Empilha(Pilha *pilha, Pedido pedido){
    if (!PilhaCheia(*pilha)){
        pilha->pedidos[pilha->topo] = pedido;
        pilha->topo++;
    }
}

bool Desempilha(Pilha *pilha){
    if (!PilhaVazia(*pilha)){
        pilha->topo--;
        return true;
    }
    return false;
}

void imprimePilha(Pilha pilha){
    for (int i = pilha.topo - 1; i >= 0; i--){
        Pedido pedido = pilha.pedidos[i];
        imprimePedido(pedido);
    }
}

void imprimePedido(Pedido pedido){
    cout << "Código pedido: " << pedido.codigo << endl;
        cout << "\tProdutos: \n";
        for (int i = 0; i < pedido.totalprodutos; i++){
            int codigoProduto = pedido.codprodutos[i];
            Produto produto = menu[codigoProduto - 1];
            cout << "\t\tCódigo: " << produto.codigo << endl;
            cout << "\t\tNome: " << produto.nome << endl;
            cout << "\t\tValor do produto: " << produto.valor << endl;
            cout << '\n';
        }
    cout << "Valor total do pedido: R$" << pedido.valorPedido << endl;
    cout << "Distância: " << pedido.distancia << " km\n";
}

int TamanhoPilha(Pilha pilha){
    return pilha.topo;
}
```

5.2. APÊNDICE B – TADs da Lista

AEDs I > TP > TP - 2 > Implementação > C++ funcoesTADs.cpp > ...

```cpp
//-----------------------------------------TADs Lista-----------------------------------------------------
void CriaListaVazia(ListaSequencial *lista){
    if (!lista->listaCriada){
        lista->listaCriada = true;
        lista->tamanho = 0;
    }
}

bool VerificaListaVazia(ListaSequencial lista){
    return lista.tamanho == 0;
}

bool VerificaListaCheia(ListaSequencial lista){
    return lista.tamanho == MAX_ENTREGA;
}

bool inserirLista(ListaSequencial *lista, Pedido pedido){
    if (!lista->listaCriada || VerificaListaCheia(*lista)) return false;
    lista->pedidos[lista->tamanho] = pedido;
    lista->tamanho++;
    return true;
}

void imprimeLista(ListaSequencial lista){
    for (int i = 0; i < lista.tamanho; i++){
        Pedido pedido = lista.pedidos[i];
        imprimePedido(pedido);
    }
}

bool RemoveIdLista(ListaSequencial *lista, int id){
    if (VerificaListaVazia(*lista))return false;
    for (int i = id; i < lista->tamanho; i++)lista->pedidos[i] = lista->pedidos[i + 1];
    lista->tamanho--;
    return true;
}

bool ListaRemove(ListaSequencial *lista, Pedido pedido){
    if (VerificaListaVazia(*lista)){
        cout << "Erro: Lista está vazia\n";
        return false;
    }
    int indice = procuraIndice(*lista, pedido);
    if (indice == -1)return false;
    return RemoveIdLista(lista, indice);
}

int procuraIndice(ListaSequencial lista, Pedido pedido){
    for (int i = 0; i < lista.tamanho; i++){
        if (lista.pedidos[i].codigo == pedido.codigo) return i;
    }
    return -1;
}

int TamanhoLista(ListaSequencial lista){
    return lista.tamanho;
}
```

5.3. APÊNDICE C – Sistema.hpp

AEDs I > TP > TP - 2 > Implementação > sistema.hpp > ...

You, há 23 horas | 1 author (You)

```hpp
#ifndef SISTEMA_H       You, ontem • Organização da Pasta do TP 2

#define SISTEMA_H
#define MAX_PRODUTOS 10
#define MAX_ENTREGA 7
#define MAX_PRODUTOS_CARDAPIO 10
#define NOME_ARQUIVO "lanchonete.bin"
#define OPCAO_SAIDA 7

#include <iostream>
#include <windows.h>
#include <fstream>

using namespace std;

int codigoPedido = 0;
```

You, há 23 horas | 1 author (You)

```hpp
typedef struct Produto{
    int codigo;
    char nome[100];
    double valor;
};

Produto menu[MAX_PRODUTOS_CARDAPIO]; // Vetor de Structs de Produtos
```

You, há 23 horas | 1 author (You)

```hpp
typedef struct Pedido{
    int codigo;
    int codprodutos[MAX_PRODUTOS];
    int totalprodutos = 0;
    double valorPedido = 0;
    double distancia = 0;
};
```

You, há 23 horas | 1 author (You)

```hpp
typedef struct Pilha{
    Pedido pedidos[MAX_ENTREGA];
    int topo;
};
```

You, há 23 horas | 1 author (You)

```hpp
typedef struct ListaSequencial{
    Pedido pedidos[MAX_ENTREGA];
    int tamanho;
    bool listaCriada = false;
};
```

```
//----------------------------------------------TADs-----------------------------------------------------------------------------------------------------

//TADs de Pilha
void InicializaPilha(Pilha *pilha);                                     // Cria a pilha
bool PilhaVazia(Pilha pilha);                                           // Verifica se a pilha está vazia
bool PilhaCheia(Pilha pilha);                                           // Verifica se a pilha está cheia
void Empilha(Pilha *pilha, Pedido pedido);                              // Adiciona item no topo
bool Desempilha(Pilha *pilha);                                          // Remove item no topo
void imprimePilha(Pilha pilha);                                         // Imprime a lista
void imprimePedido(Pedido pedido);                                      // Imprime um pedido dentro da lista
Pilha OrdenaPilha(Pilha pilha);                                         // Ordena a pilha em uma lista auxiliar e a retorna
Pedido TopoPilha(Pilha pilha);                                          // Retorna o topo da pilha
int TamanhoPilha(Pilha pilha);                                          // Tamanho da pilha

//TADs de Lista
void CriaListaVazia(ListaSequencial *lista);                            // Cria uma lista vazia
bool VerificaListaVazia(ListaSequencial lista);                         // Retorna TRUE se a lista estiver vazia, FALSE caso contrário
bool VerificaListaCheia(ListaSequencial lista);                         // Retorna TRUE se a lista estiver cheia, FALSE caso contrário
bool inserirLista(ListaSequencial *lista, Pedido pedido);               // Insere Pedido na lista ListaSequencial
void imprimeLista(ListaSequencial lista);                               // Imprime a lista
bool RemoveIdLista(ListaSequencial *lista, int id);                     // Remove um Pedido da lista via ID
bool ListaRemove(ListaSequencial *lista, Pedido pedido);                // Remove um Pedido da lista
int procuraIndice(ListaSequencial lista, Pedido pedido);                // Retorna o indice do pedido na lista, -1 se não tiver o pedido
int TamanhoLista(ListaSequencial lista);                                // Retorna o número de itens da lista


//---------------------------------------Funções do Sistema----------------------------------------------------------------------------------------------

//Arquivo
bool carregaArquivo(ListaSequencial *listaSequencial);                  //Carrega o arquivo
bool salvaArquivo(ListaSequencial *listaSequencial);                    //Salva o arquivo no fim do programa

//Formatação
void formataDecimal();                                                  // Formata os decimais deixando os números com 2 casas decimais depois da virgula

//Produtos
void adicionaCardapio(int codigo, char nome[], double valor);           //Modelo para inserir produtos
void insereProdutos();                                                  //Salvar os produtos no menu na inicialização

//Menu
void Menu();                                                            //Menu com as opções

//Funções exigidas
void incluirPedido(ListaSequencial *listaSequencial);                   // 1º - Incluir pedido na lista
bool verificaCodigoProduto(int codigoProduto);                          // Verificar se existe o código do produto que está sendo inserido no pedido
void insereProdutos(Pedido *pedido);                                    // Incluir o Produto dentro do pedido
void listarPedidos(ListaSequencial listaSequencial);                    // 2º - Lista os pedidos na ordem em foram pedidos
void verCardapio();                                                     // 3º - Mostra o cardápio
void consultarPedido(ListaSequencial listaSequencial);                  // 4º - Consulta um pedido específico com a função imprimePedido
void listaParaPilha(Pilha *pilha, ListaSequencial listaSequencial);     // Realiza a troca de lista para pilha
void imprimirListaEntrega(Pilha *pilha, ListaSequencial listaSequencial); // 5º - Após a troca de lista para pilha, ele imprime a pilha de entregas
void lancarEntrega(Pilha *pilha, ListaSequencial *listaSequencial);     // 6º - Entrega o pedido que tá no topo, que é o de menor distancia

#endif
```

5.4. APÊNDICE D – Manipulação de Arquivos

```
AEDs I > TP > TP - 2 > implementação > sistema.cpp > insereProdutos()
You, há 7 minutos | 1 author (You)
#include "sistema.hpp"

bool carregaArquivo(ListaSequencial *listaSequencial){
    ifstream arquivo;
    arquivo.open(NOME_ARQUIVO, fstream::binary); // Abre o arquivo em modo binario, ou cria se não existir

    if (arquivo.fail()){ // Se falhar
        arquivo.clear(); //Fecha
        return false;
    }

    bool primeiraLinha = true;

    while (arquivo.peek() != ifstream::traits_type::eof()){ // Enquanto o arquivo não for vazio
        if (primeiraLinha){
            arquivo >> codigoPedido;
            primeiraLinha = false;
        }
        else{
            Pedido pedido;
            arquivo.read((char *)&pedido, sizeof(Pedido)); // Lê a Struct dos pedidos
            inserirLista(listaSequencial, pedido);         // Insere o pedido que pegou do arquivo na lista sequencial
        }
    }
    arquivo.clear(); // Fecha após carregado
    return true;
}

bool salvaArquivo(ListaSequencial *listaSequencial){
    ofstream arquivo;
    arquivo.open(NOME_ARQUIVO, ofstream::trunc | fstream::binary); // Abre o arquivo em modo trunc(vazio) e em modo Binário

    if (arquivo.fail()){ // Se falhar
        arquivo.clear(); //Fecha
        return false;
    }

    arquivo << codigoPedido;

    for (int i = 0; i < listaSequencial->tamanho; i++){
        Pedido pedido = listaSequencial->pedidos[i];
        arquivo.write((char *)&pedido, sizeof(Pedido));
    }
    return true;
}
```

5.5. APÊNDICE E – Main.cpp

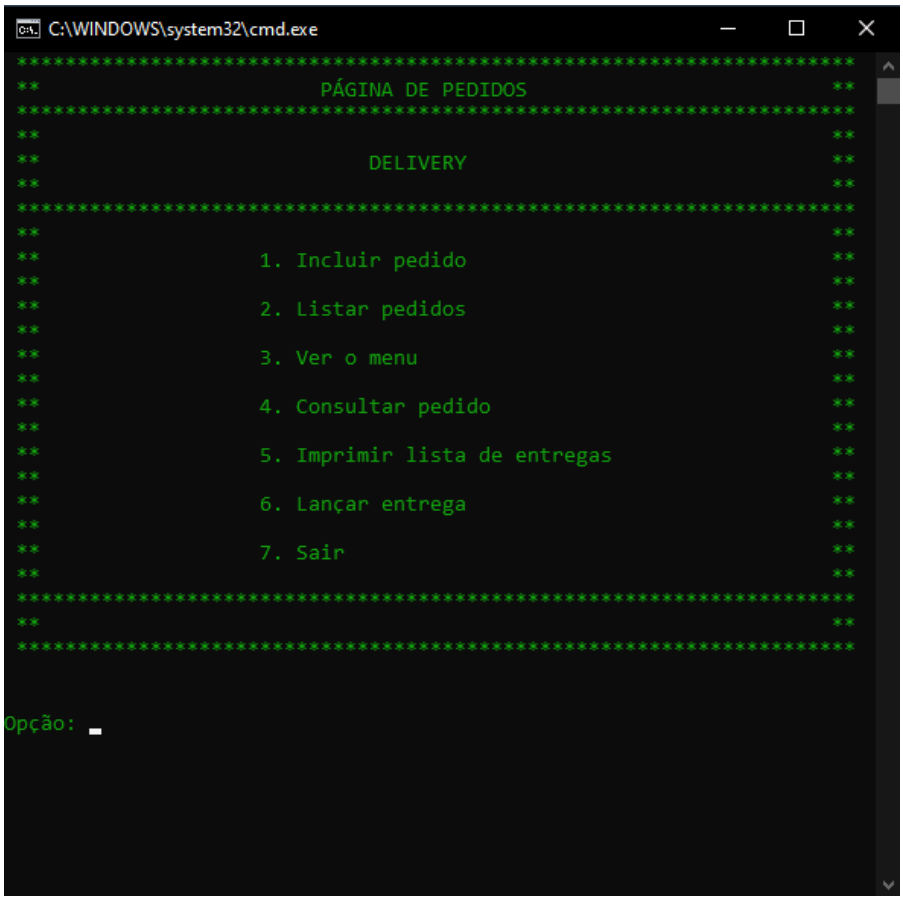
AEDs I > TP > TP - 2 > Implementação > C++ main.cpp > ...

You, há 5 minutos | 1 author (You)

```cpp
#include "sistema.hpp"

int main(){
    UINT CPAGE_UTF8 = 65001;
    UINT CPAGE_DEFAULT = GetConsoleOutputCP();
    SetConsoleOutputCP(CPAGE_UTF8);
    HANDLE colors = GetStdHandle(STD_OUTPUT_HANDLE);
    SetConsoleTextAttribute(colors, 2); // Define a cor verde para o texto

    ListaSequencial lista;
    CriaListaVazia(&lista);
    Pilha mochila;
    InicializaPilha(&mochila);

    int op;
    bool carregouComSucesso = carregaArquivo(&lista);
    if (!carregouComSucesso){
        cout << "ERRO ao carregar os Pedidos do arquivo: " << NOME_ARQUIVO;
        system("pause");
    }

    do{
        Menu();
        cin >> op;
        cin.ignore();
        system("cls");
        switch (op){
            case 1:
                incluirPedido(&lista);
                break;
            case 2:
                listarPedidos(lista);
                break;
            case 3:
                verCardapio();
                break;
            case 4:
                consultarPedido(lista);
                break;
            case 5:
                imprimirListaEntrega(&mochila, lista);
                break;
            case 6:
                lancarEntrega(&mochila, &lista);
                break;
        }
        system("cls");
    } while (op != OPCAO_SAIDA);

    bool salvouComSucesso = salvaArquivo(&lista);
    if (!salvouComSucesso)cout << "ERRO ao salvar os pedidos no arquivo: " << NOME_ARQUIVO;
    cout << "Saindo...";
    Sleep(2000);
    return 0;
}
```

5.6. APÊNDICE F – Menu

```
72  void Menu(){
73      formataDecimal();
74      insereProdutos();
75      cout << " ********************************************************************************\n";
76      cout << " **                          PÁGINA DE PEDIDOS                                 **\n";
77      cout << " ********************************************************************************\n";
78      cout << " **                                                                            **\n";
79      cout << " **                              DELIVERY                                      **\n";
80      cout << " **                                                                            **\n";
81      cout << " ********************************************************************************\n";
82      cout << " **                                                                            **\n";
83      cout << " **                    1. Incluir pedido                                       **\n";
84      cout << " **                                                                            **\n";
85      cout << " **                    2. Listar pedidos                                       **\n";
86      cout << " **                                                                            **\n";
87      cout << " **                    3. Ver o menu                                           **\n";
88      cout << " **                                                                            **\n";
89      cout << " **                    4. Consultar pedido                                     **\n";
90      cout << " **                                                                            **\n";
91      cout << " **                    5. Imprimir lista de entregas                           **\n";
92      cout << " **                                                                            **\n";
93      cout << " **                    6. Lançar entrega                                       **\n";
94      cout << " **                                                                            **\n";
95      cout << " **                    7. Sair                                                 **\n";
96      cout << " **                                                                            **\n";
97      cout << " ********************************************************************************\n";
98      cout << " **                                                                            **\n";
99      cout << " ********************************************************************************\n";
100     cout << "\n\nOpção: ";
101 }
```



5.7. APÊNDICE G – Validação do código do produto

```
103  bool verificaCodigoProduto(int codigoProduto){ // Verifica se existe produto com o código
104      if (codigoProduto < 1 || codigoProduto > MAX_PRODUTOS_CARDAPIO){
105          cout << "Não existe nenhum produto com esse código\n";
106          Sleep(1000);
107          system("cls");
108          return false;
109      }
110      return true;
111  }
```