Politécnico de Coimbra

Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Estruturas de Dados

Ano Letivo 2023/24

Trabalhos Final

Elaborado em: 2024/06/02

Guilherme Rodrigues a2020154390

Índice

1 Introdução

O objetivo é realizar um sistema em C para gerenciar o funcionamento de um café, desde a organização dos funcionários e produtos até o registro de vendas e geração de relatórios diários. O objetivo principal é garantir a eficiência operacional, o controlo do stock e a satisfação do cliente.

2 Explicação do Código

Antes de começar qualquer função foram importadas 4 bibliotecas sendo elas “stdio.h”, “stdlib.h”, “string.h” e “time.h”. Esta ultima nao foi utilizada em definitivo mas foi deixado uma função como comentario para demonstar como a utilizariamos para usar o dia atual do sistema para dar nome a um ficheiro.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
```

Figure 1: Bibliotecas utilizadas

Em seguida definimos tabem o tamanho maximo de cada tabela que vamos utilizar neste projeto. Como serão utilizadas 4 tabelas foram derinidos 4 tamanhos maximos que depois de uma análise tardia se chegou a conclusão de que talvez poderia ter definido apenas uma variavel constante.

```
#define MAX_FUNCIONARIOS 100
#define MAX_PRODUTOS 100
#define MAX_MESAS 100
#define MAX_ENCOMENDAS 100
```

Figure 2: Definição das constantes

Em seguuda criou-se as 6 estruturas necessárias. A estrutura “Funcionario”, a estrutura “Produto”, a estrutura “Encomenda”, a estrutura “IntemConta”, a estrutura “Pedido” e por fim a estrutura “Mesa”.

- Etrutura Funcionário

A estrutura “Funcionario” que irá guardar a infromação dos funcionarios. Esta estrutura guarda 3 tipos de dados, “char nome[25]”, “char funcao[25]” e “char disponibilidade[25]”. Todas possuem um tamanho fixo de 25, o ue significa que so podemos guardar até 24 caracteres. Cada uma destas varaveis irá guardar informação referente ao nome da mesma.

```
typedef struct
{
    char nome[25];
    char funcao[25];
    char disponibilidade[25];
} Funcionario;
```

Figure 3: Estrutura Funcionario

- Estrutura Produto

Esta estrutura tambem guarda 3 variaveis, “char p_nome[25]”, “float preco” e “int quantidade”. A semelhança da anterior e das restantes estruturas o nome das variáveis guardam informação referente ao nome da variavél.

```
typedef struct
{
    char p_nome[25];
    float preco;
    int quantidade;
} Produto;
```

Figure 4: Estrutura Produto

- Estrutura Encomenda

Esta estrutura guarda 2 variaveis, “char nome[25]” e “int quantidade”. É pretendido que a primeira variável guarde o nome do produto que se pretende ser encomendado e não que seja literalmetne o nome da encomenda.

```
typedef struct
{
    char nome[25];
    int quantidade;
} Encomenda;
```

Figure 5: Estrutura Encomenda

- Estrutura Item Conta

Esta estrutura para alem de 3 variáveis “char nomeProduto[25]”, “int quantidade” e “float preco” tambem possui um ponteiro para ela própria para que seja criado uma lista ligada “struct ItemConta *proximo” abrindo assim a possibilidade de guardar um número dinamico de intens sem precisar alocar previamente um grande bloco de memória.

```
typedef struct ItemConta {
    char nomeProduto[25];
    int quantidade;
    float preco;
    struct ItemConta *proximo; // Ponteiro para o próximo ItemConta na lista
} ItemConta;
```

Figure 6: Estrutura ItemConta

- Estrutura Pedido

A estrutura Pedido possui dois ponteiros, o primeiro “ItemConta *produtos” aponta para o primeiro ItemConta na lista de produtos e o segundo aponta para o proximo pedido. Assim temos uma lista de pedidos e cada pedido possui uma lista de produtos. O resto das variaveis são “int n_Mesa”, “char nomeFuncionario”, “int contadorProdutos” e “float total”.

```
typedef struct Pedido {
    int n_Mesa;
    char nomeFuncionario[25];
    ItemConta *produtos; // Ponteiro para o primeiro ItemConta na lista de produtos
    int contadorProdutos;
    float total;
    struct Pedido *proximo; // Ponteiro para o próximo Pedido na lista
} Pedido;
```

Figure 7: Estrutura Pedido

- Estrutura Mesa

Por fim a estrutura Mesa tem um ponteiro que aponta para o primeiro pedido na lista de pedidos fazendo com que cada mesa tenha uma lista de pedidos. O restante da variaveis são “int n_Mesa”, “int n_Lugares”, “char localizacao[25]” e “char m_disponibilidade”.

```
typedef struct Mesa {
    int n_Mesa;
    int n_Lugares;
    char localizacao[25];
    char m_disponibilidade[25];
    Pedido *pedidos; // Ponteiro para o primeiro pedido (Pedido) na lista de pedidos
} Mesa;
```

Figure 8: Estrutura Mesa

2.1 Funcionarios

2.1.1 Registo de Funcionarios

A afunçao “adicionarFuncionario” recebe 2 parametros “Funcionario *tabela” e “int *contador”. Estes parametros são: um ponteiro para a tabela funcionarios e um ponteiro que armazena o numero de funcionarios registados.

A primeira verificação que esta função faz é verificar se o número de funcionários (*contador) já atingiu o limite máximo (MAX_FUNCIONARIOS), caso o limite tenha sido atingido, imprime uma mensagem de erro e retorna da função.

Aloca uma variável “f” do tipo “Funcionario” e inicializa com o endereço do próximo elemento disponível na tabela “&tabela[*contador]”.

Usamos, em seguida, “scanf("%24s", f->nome)” para ler o nome do funcionário e armazená-lo em “f->nome”. O uso de “%24s” limita a entrada a 24 caracteres, evitando overflow. Repete o processo para funcao e disponibilidade.

Utilizamos um loop “while” para consumir caracteres que ainda possam estar presentes no buffer de entrada, evitando problemas na próxima leitura.

Limpamos o ecrã “system("clear")” e é exibida uma mensagem formatada com os dados do funcionário inserido (nome, funcao, disponibilidade).

Incrementamos o valor de “*contador” para indicar que um novo funcionário foi adicionado e voltamos a limpar o ecrã. No final é exibida uma mensagem pedindo ao utilizador para pressionar “Enter” para continuar. Espera a entrada do utilizador com “getchar()” antes de prosseguir.

```c
void adicionarFuncionario(Funcionario *tabela, int *contador)
{
    if (*contador >= MAX_FUNCIONARIOS)
    {
        printf("Limite de funcionarios atingido!\n");
        return;
    }

    Funcionario *f = &tabela[*contador];

    // o uso de %24s serve para evitar overflow
    // f-> serve para enviar para dentro da tabela
    printf("Insira o nome do funcionario: ");
    scanf("%24s", f->nome);
    printf("Insira a função do funcionario: ");
    scanf("%24s", f->funcao);
    printf("Insira a disponibilidade atual do funcionario: ");
    scanf("%24s", f->disponibilidade);

    // Limpamos o buffer
    int c;
    while ((c = getchar()) != '\n' && c != EOF) { }

    printf("\nPressione Enter para mostrar os dados do funcionario inserido...\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");

    printf("\n-----------------Funcionario adicionado:-----------------\n");
    printf("Nome: %s\n", f->nome);
    printf("Função: %s\n", f->funcao);
    printf("Disponibilidade: %s\n", f->disponibilidade);
    printf("\n---------------------------------------------------------------\n");

    (*contador)++;

    printf("\nPressione Enter para continuar\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");

}
```

Figure 9: Função para adicionar funcionários

2.1.2 Mostrar Funcionários

A função “mostraFuncionarios” recebe tambem dois parametros, recebe o ponteiro para a tabela de funcionários e uma variável que armazena o número de funcionários registados.

Primeiramente exibimos um cabeçalho formatado com o título "Lista de Funcionarios Registados".

Depois usamos um loop for para percorrer a tabela de funcionários do primeiro ao último elemento (até contador - 1).

Para cada iteração obtemos a referência do funcionário atual “Funcionario *f = &tabela[i]”, exibmos o número do funcionário formatado (por exemplo, "Funcionario 1", "Funcionario 2") e Imprime os dados do funcionário: nome, função e disponibilidade, utilizando “printf()” e “f->nome”, “f->funcao” e “f->disponibilidade”.

Por fim limpamos o buffer e pedimos ao utilizador que prima “Enter” para avançar como na função anterior.

```c
void mostrarFuncionarios(Funcionario *tabela, int contador)
{

    printf("\nLista de Funcionarios Registados:\n");
    printf("------------------------------------------------------------\n");
    for (int i = 0; i < contador; i++)
    {
        Funcionario *f = &tabela[i];
        printf("Funcionario %d:\n", i + 1);
        printf("Nome: %s\n", f->nome);
        printf("Função: %s\n", f->funcao);
        printf("Disponibilidade: %s\n", f->disponibilidade);
        printf("------------------------------------------------------------\n");
    }

    int c;
    while ((c = getchar()) != '\n' && c != EOF) { }
    printf("\n\nPressione Enter para continuar...\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");
}
```

Figure 10: Função para mostrar funcionarios registados

2.1.3 Alterar Funcionários

Esta função recebe os mesmo parametros da anterior.

Primeiro pedimos ao utilizador que insira o nome do funcionário que pretende alterar usando as funções

“printf()” e “scanf()”. Em seguida usamos um loop for para percorrer a tabela de funcionários do primeiro ao último elemento (até contador - 1). Para cada iteração comparamos o nome do funcionário atual “tabela[i].nome” com o nome inserido pelo utilizador “nomeFuncionario”. Se os nomes forem iguais a variável “funcionarioEncontrado” é definida como 1, indicando que o funcionário foi encontrado. Armazenamos a referência do funcionário encontrado em “f (&tabela[i])”.e saimos do loop for usando break.

Se o loop terminar sem encontrar o funcionário: Informa ao utilizador que o funcionário não foi encontrado “printf("Funcionário %s não encontrado.\n", nomeFuncionario);”.

```c
void alterarFuncionarios(Funcionario *tabela, int contador)
{
    int opcao;
    char nomeFuncionario[25];
    int funcionarioEncontrado = 0;

    printf("insira o nome do funcionário que pretende alterar: ");
    scanf("%24s", nomeFuncionario);

    // Verificar existência do funcionário
    Funcionario *f = NULL;
    for (int i = 0; i < contador; i++)
    {
        if (strcmp(tabela[i].nome, nomeFuncionario) == 0)
        {
            funcionarioEncontrado = 1;
            f = &tabela[i];
            break;
        }
    }

    if (!funcionarioEncontrado)
    {
        printf("Funcionário %s não encontrado.\n", nomeFuncionario);
        return;
    }

    // Atualizar as informações do funcionário
    printf("Insira a nova função do funcionário: ");
    scanf("%24s", f->funcao);
    printf("Insira a nova disponibilidade do funcionário: ");
    scanf("%24s", f->disponibilidade);
```

Figure 11: Função para alterar Funcionarios 1

```c
// Limpar o buffer de entrada
int c;
while ((c = getchar()) != '\n' && c != EOF) {}

printf("\nPressione Enter para mostrar os dados do funcionário alterado...\n");
getchar(); // Espera que o utilizador pressione Enter
system("clear");

printf("\n------------------Funcionário alterado:------------------\n");
printf("Nome: %s\n", f->nome);
printf("Função: %s\n", f->funcao);
printf("Disponibilidade: %s\n", f->disponibilidade);
printf("\n-----------------------------------------------------------\n");

printf("\nPressione Enter para continuar\n");
getchar(); // Espera que o utilizador pressione Enter
system("clear");
```

Figure 12: Função para alterar Funcionarios 2

2.1.4 Remover Funcionários

Tambem recebe os mesmo parametros da função de registo.

Começamos tambem por pedir ao utilizador o nome do funcionário que é pretendido remover usando as funções “printf()” e “scanf()”.

Usamos novamente um loop “for” para percorrer a tabela de funcionários do primeiro ao último elemento (até *contador - 1). Para cada iteração voltamos a comparar o nome do funcionário atual “tabela[i].nome” com o nome inserido pelo utilizador “nomeFuncionario”. Se os nomes forem iguais definimos a variável “funcionarioEncontrado” como 1, indicando que o funcionário foi encontrado.

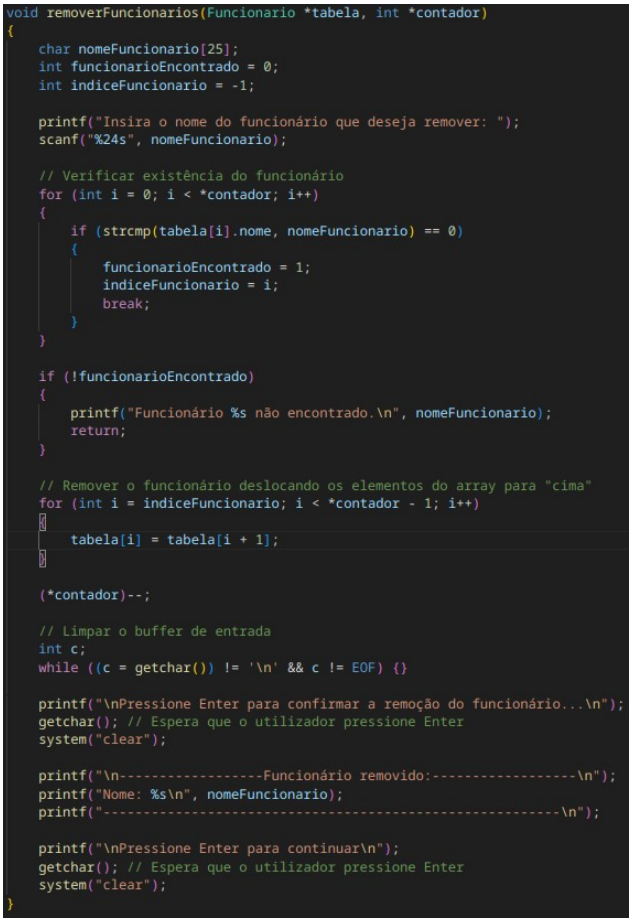
Armazenamos o índice do funcionário encontrado em “indiceFuncionario (I)” e saimos do loop for usando “break”.

Se o loop for terminar sem encontrar o funcionário informamos o utilizador que o funcionário não foi encontrado “printf("Funcionário %s não encontrado.\n", nomeFuncionario);”

Se o funcionário for encontrado utilizamos um loop for a partir do índice do funcionário encontrado “indiceFuncionario” até ao penúltimo elemento da tabela “*contador - 2”.

Para cada iteração são copiados os dados do funcionário seguinte “tabela[i + 1]” para o elemento atual “tabela[i]” e decrementamos o valor de “*contador” para indicar que há um funcionário a menos na tabela.

Por fim assim como nas anteriores e em todas as proximas limpamos o buffer e esperamos o “Enter” do utilizador.



```c
void removerFuncionarios(Funcionario *tabela, int *contador)
{
    char nomeFuncionario[25];
    int funcionarioEncontrado = 0;
    int indiceFuncionario = -1;

    printf("Insira o nome do funcionário que deseja remover: ");
    scanf("%24s", nomeFuncionario);

    // Verificar existência do funcionário
    for (int i = 0; i < *contador; i++)
    {
        if (strcmp(tabela[i].nome, nomeFuncionario) == 0)
        {
            funcionarioEncontrado = 1;
            indiceFuncionario = i;
            break;
        }
    }

    if (!funcionarioEncontrado)
    {
        printf("Funcionário %s não encontrado.\n", nomeFuncionario);
        return;
    }

    // Remover o funcionário deslocando os elementos do array para "cima"
    for (int i = indiceFuncionario; i < *contador - 1; i++)
    {
        tabela[i] = tabela[i + 1];
    }

    (*contador)--;

    // Limpar o buffer de entrada
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {}

    printf("\nPressione Enter para confirmar a remoção do funcionário...\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");

    printf("\n------------------Funcionário removido:------------------\n");
    printf("Nome: %s\n", nomeFuncionario);
    printf("---------------------------------------------------------\n");

    printf("\nPressione Enter para continuar\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");
}
```

Figure 13: Função para remover um funcionario

2.1.5 Guardar funcionários

Esta função recebe os parametros iguais aos da função de mostragem.

Utilizamos a função “fopen()” para abrir o arquivo "funcionarios.dat" no modo binário de escrita ("wb"). Se a abertura do arquivo falhar “file == NULL” exibimos uma mensagem de erro informando que o arquivo não pôde ser aberto e retornamos da função “return;”.

Fazemos uso da função “fwrite()” para escrever o valor da variável “contador” no arquivo. Os argumentos de “fwrite()” são: “&contador”, “sizeof(int)”, “1” e “file”.

Em seguida utilizamos a função “fwrite()” novamente para escrever os dados da tabela de funcionários “f_tabela” no arquivo. Os argumentos de “fwrite()”são: “f_tabela”, “sizeof(Funcionario)”, “contador” e “file”.

Por fim usamos a função “fclose()” para fechar o arquivo.

```c
void salvarFuncionarios(Funcionario *f_tabela, int contador)
{
    FILE *file = fopen("funcionarios.dat", "wb");
    if (file == NULL)
    {
        printf("Erro ao abrir arquivo para salvar funcionários.\n");
        return;
    }
    fwrite(&contador, sizeof(int), 1, file);
    fwrite(f_tabela, sizeof(Funcionario), contador, file);
    fclose(file);
}
```

Figure 14: Função para guardar funcionarios num ficheiro Binario

2.1.6 Carregar Funcionários

Mais uma vez utilizamos a função “fopen()” para abrir o arquivo "funcionarios.dat" no modo binário de leitura ("rb"). Se a abertura do arquivo falhar “file == NULL”, exibe uma mensagem de erro informando que o arquivo não pôde ser aberto.

Seguidamente a função fread() para ler o valor do contador de funcionários do arquivo. Os argumentos de “fread()” são: “&contador”, “sizeof(int)”, “1”e “file”.

Mais uma vez usamos a função “fread()” para ler os dados da tabela de funcionários “f_tabela” do arquivo. Os argumentos de fread() são: “f_tabela”, “sizeof(Funcionario)”, ”contador” e “file”.

Por fim fechamos o arquivo com o uso da função “fclose()”.

```c
int carregarFuncionarios(Funcionario *f_tabela)
{
    FILE *file = fopen("funcionarios.dat", "rb");
    if (file == NULL) {
        printf("Erro ao abrir arquivo para carregar funcionários.\n");
        return 0;
    }
    int contador;
    fread(&contador, sizeof(int), 1, file);
    fread(f_tabela, sizeof(Funcionario), contador, file);
    fclose(file);
    return contador;
}
```

Figure 15: Função para carregar a informação do arquivo binário para a memória

2.1.7 Menu dos Funcionario

A função para o menu dos funcionários é apenas um loop “while” que com “switch case” para o utilizador poder chamar chamar as funções de acordo com o numero da opção escolhida.

```c
void menuFuncionarios(Funcionario *tabela, int *contador)
{
    int opcao;

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Inserir funcionário\n");
        printf("2. Ver funcionarios\n");
        printf("3. Alterar funcionário\n");
        printf("4. Remover funcionário\n");
        printf("5. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        // Limpar o buffer de entrada
        int c;

        switch (opcao)
        {
            case 1:
                system("clear");
                adicionarFuncionario(tabela, contador);
                break;
            case 2:
                system("clear");
                mostrarFuncionarios(tabela, *contador);
                break;
            case 3:
                system("clear");
                alterarFuncionarios(tabela, *contador);
                break;
            case 4:
                system("clear");
                removerFuncionarios(tabela, contador);
                break;
            case 5:
                system("clear");
                printf("Saindo...\n");
                return;
            default:
                printf("Opção inválida. Tente novamente.\n");
        }
    }
}
```

Figure 16: Menu dos Funcionários

2.2 Produtos

2.2.1 Adicionar produtos

Esta função recebe 2 parametros:

- “p_tabela”: Ponteiro para a tabela de produtos (array de structs Produto).
- “contador”: Ponteiro para a variável que armazena o número de produtos registados.

Primeiro a função verifica se o número de produtos registados “*contador” já atingiu o limite máximo “MAX_PRODUTOS”. Se o limite for atingido, exibe uma mensagem de erro e retorna da função.

Em segundo cria uma variável temporária “novoProduto” do tipo “Produto” para armazenar os dados do novo produto. Depois pede ao utilizador que insira o nome do produto “novoProduto.p_nome” utilizando “scanf("%24s", novoProduto.p_nome)”.

A seguir pede ao utilizador que insira o preço do produto “novoProduto.preco” utilizando “scanf("%f", &(novoProduto.preco)”.

Limpamos o buffer de entrada “while ((c = getchar()) != '\n' && c != EOF) { }” para evitar problemas na próxima leitura e pedimos ao utilizador que insira a quantidade inicial do produto “novoProduto.quantidade” utilizando “scanf("%d", &(novoProduto.quantidade)”.

Em terceiro é definida a variável “posicao” para o valor atual do contador “*contador”, indicando a última posição na tabela.

Utilizamos um loop “for” para percorrer a tabela de produtos do início ao fim “i < *contador” e dentro desse mesmo loop, comparamos a quantidade do novo produto “novoProduto.quantidade” com a quantidade de cada produto na tabela “p_tabela[i].quantidade”. Se a quantidade do novo produto for menor que a quantidade de algum produto na tabela, atualiza a variável “posicao” para a posição desse produto (para manter a ordenação por quantidade).

Em quarto utilizamos um loop “for” a partir da última posição na tabela “*contador” até a posição de inserção “posicao”. Ainda dentro do loop, copiamos os dados do produto na posição anterior “p_tabela[i - 1]” para a posição atual “p_tabela[i]”.

Em quinto é inserida a variável “novoProduto” na posição posicao da tabela de produtos “p_tabela[posicao] = novoProduto”.

Em sexto incrementamos o valor da variável “*contador” para indicar que há um novo produto na tabela.

Por fim mostramos o produto inserido.

```c
void adicionarProduto(Produto *p_tabela, int *contador) //como pensei que era para
{
    if (*contador >= MAX_PRODUTOS)
    {
        printf("Limite de produtos atingido!\n");
        return;
    }

    Produto novoProduto;

    // o uso de %24s serve para evitar overflow
    printf("Insira o nome do produto: ");
    scanf("%24s", novoProduto.p_nome);
    printf("Insira o preço do produto: ");
    scanf("%f", &(novoProduto.preco));
    int c;
    while ((c = getchar()) != '\n' && c != EOF) { }
    printf("Insira a quantidade atual do produto: ");
    scanf("%d", &(novoProduto.quantidade));

    // Limpamos o buffer
    while ((c = getchar()) != '\n' && c != EOF) { }

    // Encontrar a posição correta para inserir o novo produto
    int posicao = *contador;
    for (int i = 0; i < *contador; i++)
    {
        if (novoProduto.quantidade < p_tabela[i].quantidade)
        {
            posicao = i;
            break;
        }
    }

    // Deslocar os elementos para a direita para abrir espaço para o novo produto
    for (int i = *contador; i > posicao; i--)
    {
        p_tabela[i] = p_tabela[i - 1];
    }

    // Inserir o novo produto na posição correta
    p_tabela[posicao] = novoProduto;

    (*contador)++;

    system("clear");
    printf("\nPressione Enter para mostrar os dados do produto inserido...\n");
    getchar(); // Espera que o utilizador pressione Enter
```

Figure 17: Função para registar produtos 1

```c
    printf("\n------------------Produto adicionado:------------------\n");
    printf("Nome: %s\n", novoProduto.p_nome);
    printf("Preço: %0.2f\n", novoProduto.preco);
    printf("Quantidade: %d\n", novoProduto.quantidade);
    printf("\n-------------------------------------------------------------\n");

    printf("\nPressione Enter para continuar...\n");
    getchar(); // Espera que o utilizador pressione Enter
}
```

Figure 18: Função para registar produtos 2

2.2.2 Mostrar produtos

Exibimos uma mensagem a pedir ao utilizador que escolha o critério de exibição dos produtos. Esses critérios são: Quantidade (padrão) e Preço.

Se o utilizador escolher o critério de ordenação por preço “criterio == 2”: Éu utilizado o algoritmo de ordenação bubble sort para ordenar os produtos por preço em ordem crescente.

O algoritmo funciona da seguinte maneira: Faz um loop “for” externo que percorre a tabela de produtos do início ao fim “i = 0; i < contador - 1; i++”. Dentro do loop externo, faz um loop “for” interno que percorre a tabela do início ao fim “j = 0; j < contador - i - 1; j++”. Compara o preço do produto na posição atual “p_tabela[j].preco” com o preço do produto na próxima posição “p_tabela[j + 1].preco”. Se o preço do produto na posição atual for maior que o preço do produto na próxima posição: Troca os produtos de lugar utilizando uma variável temporária “temp”. Essa troca coloca o produto com preço menor na posição anterior e o produto com preço maior na posição posterior, aproximando a ordenação.

Por fim exibimos o resultado.

```c
void mostrarProdutos(Produto *p_tabela, int contador)
{
 int criterio;

    printf("Escolha o critério de exibição:\n");
    printf("1. Quantidade (padrão)\n");
    printf("2. Preço\n");
    printf("Digite sua escolha: ");
    scanf("%d", &criterio);

    // Limpar o buffer de entrada
    int c;
    while ((c = getchar()) != '\n' && c != EOF) { }

    if (criterio == 2)
    {
        // Ordenar os produtos por preço usando bubble sort
        for (int i = 0; i < contador - 1; i++)
        {
            for (int j = 0; j < contador - i - 1; j++)
            {
                if (p_tabela[j].preco > p_tabela[j + 1].preco)
                {
                    Produto temp = p_tabela[j];
                    p_tabela[j] = p_tabela[j + 1];
                    p_tabela[j + 1] = temp;
                }
            }
        }
    }

    printf("\nLista de Produtos Registrados:\n");
    printf("-------------------------------------------------------------\n");
    for (int i = 0; i < contador; i++)
    {
        Produto *produto = &p_tabela[i];
        printf("Produto %d:\n", i + 1);
        printf("Nome: %s\n", produto->p_nome);
        printf("Preço: %.2f\n", produto->preco);
        printf("Quantidade: %d\n", produto->quantidade);
        printf("-------------------------------------------------------------\n");
    }

    while ((c = getchar()) != '\n' && c != EOF) { }
    printf("\n\nPressione Enter para continuar...\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");
}
```

Figure 19: Função para mostrar os produtos registados por uma ordem a escolha do utilizador

2.2.3 Alterar produtos

Começamos por pedir ao utilizador o nome do produto que pretende alterar usando “scanf("%24s", nomeProduto)”.

Utilizamos um loop “for” para percorrer a tabela de produtos do início ao fim “i < contador”. Para cada produto: Compara o nome do produto atual “p_tabela[i].p_nome” com o nome digitado pelo usuário “nomeProduto”. Se os nomes forem iguais: Define a variável “produtoEncontrado” como 1, indicando que o produto foi encontrado. Obtém um ponteiro para o produto encontrado “Produto *p = &p_tabela[i]”.

Se o produto foi encontrado “produtoEncontrado == 1”: Pede ao utilizar que digite o novo nome do produto “printf("Insira o novo nome do produto: ")”; “scanf("%24s", p->p_nome);”. Pedimos ao utilizador que digite o novo preço do produto “printf("Insira o novo preço do produto: ")”; “scanf("%f", &(p->preco)”. Limpamos o buffer de entrada. Pedimos ao utilizador que digite a nova quantidade do produto “printf("Insira a nova quantidade do produto: ")”; “scanf("%d", &(p->quantidade))”.

Por fim apresentamos o produto alterado.

```c
void alterarProdutos(Produto *p_tabela, int contador)
{
    char nomeProduto[25];
    int produtoEncontrado = 0;

    printf("Insira o nome do produto que deseja alterar: ");
    scanf("%24s", nomeProduto);

    // Verificar existência do produto
    for (int i = 0; i < contador; i++)
    {
        if (strcmp(p_tabela[i].p_nome, nomeProduto) == 0)
        {
            produtoEncontrado = 1;
            Produto *p = &p_tabela[i];

            printf("Insira o novo nome do produto: ");
            scanf("%24s", p->p_nome);
            printf("Insira o novo preço do produto: ");
            scanf("%f", &(p->preco));
            int c;
            while ((c = getchar()) != '\n' && c != EOF) { }
            printf("Insira a nova quantidade do produto: ");
            scanf("%d", &(p->quantidade));

            // Limpamos o buffer
            while ((c = getchar()) != '\n' && c != EOF) { }

            system("clear");
            printf("\nPressione Enter para mostrar os dados do produto alterado...\n");
            getchar(); // Espera que o utilizador pressione Enter

            printf("\n-----------------Produto alterado:-----------------\n");
            printf("Nome: %s\n", p->p_nome);
            printf("Preço: %0.2f\n", p->preco);
            printf("Quantidade: %d\n", p->quantidade);
            printf("\n---------------------------------------------------------------\n");

            break;
        }
    }

    if (!produtoEncontrado)
    {
        printf("Produto %s não encontrado.\n", nomeProduto);
    }

    printf("\nPressione Enter para continuar\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");
}
```

Figure 20: Função para alterar os produtos

2.2.4 Remover Produtos

Mais uma vez pedimos ao utilizador que insira o nome do produto que pretende remover usando as funções “printf()” e “scanf()”.

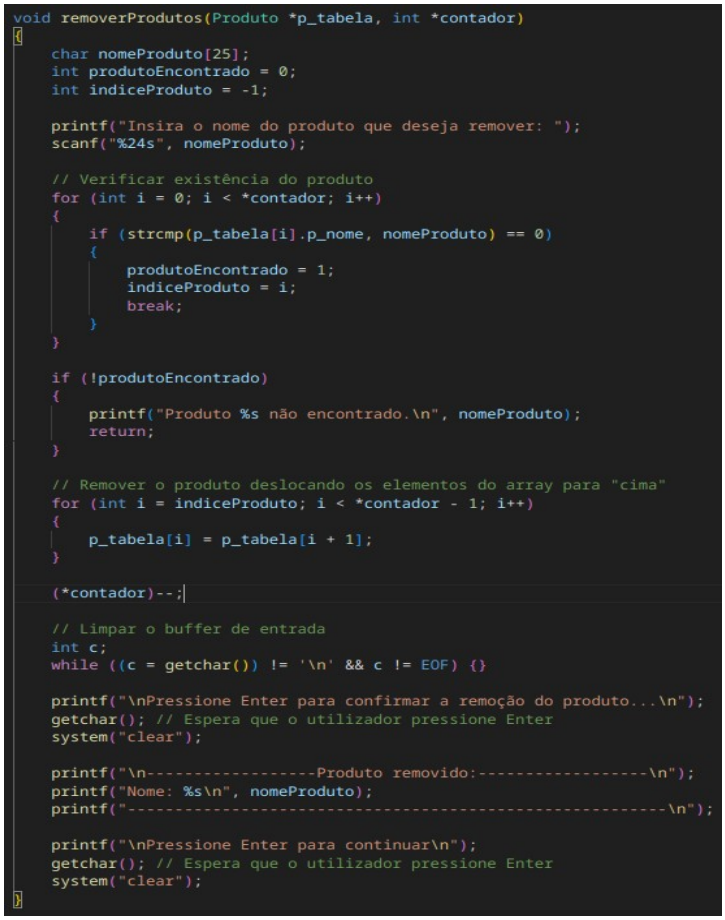
Utilizamos um loop “for” para percorrer a tabela de produtos do início ao fim “i < *contador”. Para cada produto: Compara o nome do produto atual “p_tabela[i].p_nome” com o nome inserido pelo utilizador “nomeProduto”. Se os nomes forem iguais: Define a variável “produtoEncontrado” como “1”, indicando que o produto foi encontrado. Armazena o índice do produto na variável “indiceProduto”.

Se o produto não for encontrado “!produtoEncontrado”, 2 exibida uma mensagem informando que o produto não foi localizado e retorna da função (return;).

Se o produto foi encontrado “produtoEncontrado == 1”: Utilizamos um loop “for” a partir do índice do produto encontrado “indiceProduto” até o último índice da tabela “*contador - 1”. Dentro do loop, copia o produto da próxima posição para a posição atual, deslocando os produtos "para cima" da tabela. Esse deslocamento remove o produto original da tabela.

A seguir decrementamos o valor da variável “*contador” para indicar que há um produto a menos na tabela.

Por fim limpamos o buffer e esperamos o “Enter” pelo utilizador e apresentamos o produto removido.

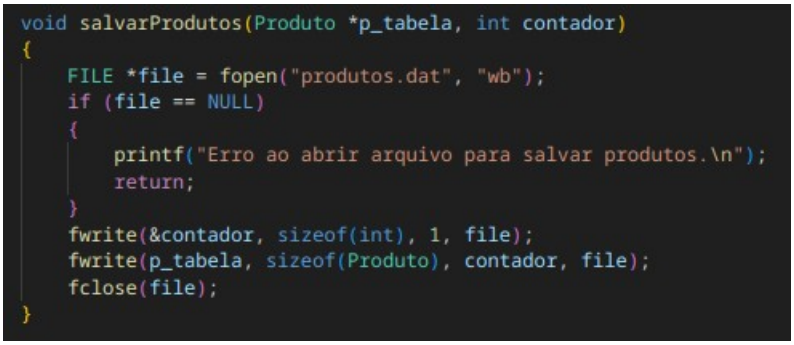


```c
void removerProdutos(Produto *p_tabela, int *contador)
{
    char nomeProduto[25];
    int produtoEncontrado = 0;
    int indiceProduto = -1;

    printf("Insira o nome do produto que deseja remover: ");
    scanf("%24s", nomeProduto);

    // Verificar existência do produto
    for (int i = 0; i < *contador; i++)
    {
        if (strcmp(p_tabela[i].p_nome, nomeProduto) == 0)
        {
            produtoEncontrado = 1;
            indiceProduto = i;
            break;
        }
    }

    if (!produtoEncontrado)
    {
        printf("Produto %s não encontrado.\n", nomeProduto);
        return;
    }

    // Remover o produto deslocando os elementos do array para "cima"
    for (int i = indiceProduto; i < *contador - 1; i++)
    {
        p_tabela[i] = p_tabela[i + 1];
    }

    (*contador)--;

    // Limpar o buffer de entrada
    int c;
    while ((c = getchar()) != '\n' && c != EOF) {}

    printf("\nPressione Enter para confirmar a remoção do produto...\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");

    printf("\n------------------Produto removido:------------------\n");
    printf("Nome: %s\n", nomeProduto);
    printf("-----------------------------------------------------------\n");

    printf("\nPressione Enter para continuar\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");
}
```

Figure 21: Função para remover produtos

2.2.5 Salvar produtos

Esta função é igual á função de guardar de funcionários apenas difere em algumas variáveis.



```c
void salvarProdutos(Produto *p_tabela, int contador)
{
    FILE *file = fopen("produtos.dat", "wb");
    if (file == NULL)
    {
        printf("Erro ao abrir arquivo para salvar produtos.\n");
        return;
    }
    fwrite(&contador, sizeof(int), 1, file);
    fwrite(p_tabela, sizeof(Produto), contador, file);
    fclose(file);
}
```

Figure 22: Função para salvar os produtos

2.2.6 Carregar produtos

Esta função é igual ao carregamento de funcionários apenas difere em algumas variáveis.

```c
int carregarProdutos(Produto *p_tabela)
{
    FILE *file = fopen("produtos.dat", "rb");
    if (file == NULL)
    {
        printf("Erro ao abrir arquivo para carregar produtos.\n");
        return 0;
    }
    int contador;
    fread(&contador, sizeof(int), 1, file);
    fread(p_tabela, sizeof(Produto), contador, file);
    fclose(file);
    return contador;
}
```

Figure 23: Função para carregar os produtos

2.2.7 Menu dos Produtos

Esta função é igual ao menu dos funcionarios apenas difere nas funções que são chamadas.

```c
> int carregarProdutos(Produto *p_tabela) ···

void menuProdutos(Produto *p_tabela, int *contador)
{
    int opcao;

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Inserir Produto\n");
        printf("2. Mostrar Produtos\n");
        printf("3. Alterar Produtos\n");
        printf("4. Remover Produtos\n");
        printf("5. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);
        getchar(); // Limpa o buffer de entrada após ler um número

        switch (opcao)
        {
            case 1:
                adicionarProduto(p_tabela, contador);
                system("clear");
                break;
            case 2:
                mostrarProdutos(p_tabela, *contador);
                system("clear");
                break;
            case 3:
                alterarProdutos(p_tabela, *contador);
                system("clear");
                break;
            case 4:
                removerProdutos(p_tabela, contador);
                system("clear");
                break;
            case 5:
                printf("Saindo...\n");
                system("clear");
                return;
            default:
                printf("Opção inválida. Tente novamente.\n");
        }
    }
}
```

Figure 24: Função de Menu para os Produtos

2.3 Mesas

2.3.1 Adicionar uma mesa

A semelhança das demais verificamos se o número de mesas cadastradas “*contador” já atingiu o limite máximo “MAX_MESAS”. Se o limite for atingido, exibe uma mensagem de erro e retorna da função.

Em seguida obtemos um ponteiro para a nova mesa a ser adicionada na última posição da tabela “f = &m_tabela[*contador]”.

Pedimos ao utilizador que digite o número da mesa “printf("Insira o numero da mesa: ")”; “scanf("%d", &(f->n_Mesa));”. Em seguida pedimos ao utilizador que insira o número de lugares da mesa “printf("Insira o numero de lugares da mesa: ")”; “scanf("%d", &(f->n_Lugares))”. Depois pedimos ao usuário que insira a localização da mesa “printf("Insira a localizacao da mesa: ")”; “scanf("%24s", f->localizacao);”. Por fim so falta perdir ao utilizador que diga a disponibilidade atual da mesa “printf("Insira a disponibilidade atual da mesa: ")”; “scanf("%24s", f->m_disponibilidade)”.

Inicializamos a lista de pedidos da mesa como “NULL” “f->pedidos = NULL;”. Isso indica que a nova mesa ainda não possui nenhum pedido associado a ela.

Por fim mostramos a mesa inserida e incrementamos a variavel “*contador”.

```c
void adicionarMesa(Mesa *m_tabela, int *contador)
{
    if (*contador >= MAX_MESAS)
    {
        printf("Limite de mesas atingido!\n");
        return;
    }

    Mesa *f = &m_tabela[*contador];

    printf("Insira o numero da mesa: ");
    scanf("%d", &(f->n_Mesa));
    printf("Insira o numero de lugares da mesa: ");
    scanf("%d", &(f->n_Lugares));
    printf("Insira a localizacao da mesa: ");
    scanf("%24s", f->localizacao);
    printf("Insira a disponibilidade atual da mesa: ");
    scanf("%24s", f->m_disponibilidade);

    // Inicializa a lista de pedidos como NULL
    f->pedidos = NULL;

    // Limpar o buffer
    int c;
    while ((c = getchar()) != '\n' && c != EOF) { }

    system("clear");
    printf("\nPressione Enter para mostrar os dados da mesa inserida...\n");
    getchar(); // Espera que o utilizador pressione Enter

    printf("\n------------------Mesa adicionada:------------------\n");
    printf("Numero: %d\n", f->n_Mesa);
    printf("Numero de lugares: %d\n", f->n_Lugares);
    printf("Localizacao: %s\n", f->localizacao);
    printf("Disponibilidade: %s\n", f->m_disponibilidade);
    printf("\n----------------------------------------------------------\n");

    (*contador)++;

    printf("\nPressione Enter para continuar...\n");
    getchar(); // Espera que o utilizador pressione Enter
}
```

Figure 25: Função para inserir uma nova mesa

2.3.2 Mostrar mesa

Começamos por pedir ao utilizador o numero da mesa que pretende observar.

Utilizamos um loop “for” para percorrer a tabela de mesas do início ao fim “i < contadorMesas”. Para cada mesa, depois comparamos o número da mesa atual “m_tabela[i].n_Mesa” com o número da mesa digitado pelo usuário “mesaEscolhida”. Se os números coincidirem define a variável “mesaEncontrada” como “1”, indicando que a mesa foi encontrada. Obtemos um ponteiro para a mesa atual “Mesa*mesaAtual = &m_tabela[i]”. Obtém um ponteiro para o primeiro pedido da mesa “Pedido *pedidoAtual = mesaAtual->pedidos”.

Se a lista de pedidos da mesa estiver vazia “pedidoAtual == NULL” e exibe uma mensagem informando que não há pedidos para a mesa e retorna da função “return;”.

Se a mesa foi encontrada e possui pedidos “mesaEncontrada == 1 e pedidoAtual != NULL” exibe um cabeçalho formatado para os pedidos da mesa “printf("\nPedidos da Mesa %d:\n", mesaEscolhida)”. Percorre a lista de pedidos da mesa utilizando um loop “while (pedidoAtual != NULL)”.

Para cada pedido é exibido o número da mesa do pedido “printf("Pedido referente à Mesa: %d\n", pedidoAtual->n_Mesa)”, o nome do funcionário que registrou o pedido “printf("Nome do Funcionário: %s\n", pedidoAtual->nomeFuncionario)”, o total de produtos no pedido “printf("Total de Produtos: %d\n", pedidoAtual->contadorProdutos)”, o total do valor do pedido formatado com duas casas decimais “printf("Total do Pedido: %.2f\n", pedidoAtual->total)” .

Depois percorre a lista de itens do pedido utilizando um loop “while (itemAtual != NULL)”.

Para cada item é exibido um cabeçalho para o item “printf(" Produto:\n")”, o nome do produto “printf(" Nome: %s\n", itemAtual->nomeProduto)”, a quantidade do produto “printf(" Quantidade: %d\n", itemAtual->quantidade)”, o preço do produto formatado com duas casas decimais “printf(" Preço: %.2f\n", itemAtual->preco)” e passa para o próximo item na lista “itemAtual = itemAtual->proximo;”.

```c
void mostrarMesas(Mesa *m_tabela, int contadorMesas)
{
int mesaEscolhida;
    int mesaEncontrada = 0;

    printf("Selecione a mesa para ver os pedidos: ");
    scanf("%d", &mesaEscolhida);

    for (int i = 0; i < contadorMesas; i++)
    {
        if (m_tabela[i].n_Mesa == mesaEscolhida)
        {
            mesaEncontrada = 1;
            Mesa *mesaAtual = &m_tabela[i];
            Pedido *pedidoAtual = mesaAtual->pedidos;

            if (pedidoAtual == NULL)
            {
                printf("Não há pedidos para a mesa %d.\n", mesaEscolhida);
                return;
            }

            printf("\nPedidos da Mesa %d:\n", mesaEscolhida);
            printf("------------------------------------------------------------\n");

            while (pedidoAtual != NULL)
            {
                printf("Pedido referente à Mesa: %d\n", pedidoAtual->n_Mesa);
                printf("Nome do Funcionário: %s\n", pedidoAtual->nomeFuncionario);
                printf("Total de Produtos: %d\n", pedidoAtual->contadorProdutos);
                printf("Total do Pedido: %.2f\n", pedidoAtual->total);

                ItemConta *itemAtual = pedidoAtual->produtos;
                while (itemAtual != NULL)
                {
                    printf("  Produto:\n");
                    printf("    Nome: %s\n", itemAtual->nomeProduto);
                    printf("    Quantidade: %d\n", itemAtual->quantidade);
                    printf("    Preço: %.2f\n", itemAtual->preco);
                    itemAtual = itemAtual->proximo;
                }

                printf("------------------------------------------------------------\n");
                pedidoAtual = pedidoAtual->proximo;
            }
        }
    }

    if (!mesaEncontrada)
    {
        printf("Mesa %d não encontrada.\n", mesaEscolhida);
    }

    int c;
    while ((c = getchar()) != '\n' && c != EOF) { }
    printf("\n\nPressione Enter para continuar...\n");
    getchar(); // Espera que o utilizador pressione Enter
    system("clear");
}
```

Figure 26: Função para exibir as mesas e os seus pedidos

2.3.3 Salvar mesas

Esta função é igual á função de guardar de funcionários apenas difere em algumas variáveis.

```c
void salvarMesas(Mesa *m_tabela, int contador)
{
    FILE *file = fopen("mesas.dat", "wb");
    if (file == NULL)
    {
        printf("Erro ao abrir arquivo para salvar mesas.\n");
        return;
    }
    fwrite(&contador, sizeof(int), 1, file);
    fwrite(m_tabela, sizeof(Mesa), contador, file);
    fclose(file);
}
```

Figure 27: Função para guardar as mesas num ficheiro binário

2.3.4 Carregar mesas

Esta função é igual ao carregamento de funcionários apenas difere em algumas variáveis.

```c
int carregarMesas(Mesa *m_tabela)
{
    FILE *file = fopen("mesas.dat", "rb");
    if (file == NULL)
    {
        printf("Erro ao abrir arquivo para carregar mesas.\n");
        return 0;
    }
    int contador;
    fread(&contador, sizeof(int), 1, file);
    fread(m_tabela, sizeof(Mesa), contador, file);
    fclose(file);
    return contador;
}
```

Figure 28: Função para carregar as mesas do ficheiro binário

2.3.5 Abrir um pedido/conta

Nesta função teremos de associar um funcionario e registar os pedidos, assim esta função recebe os parametros:

- “m_tabela”: Ponteiro para a tabela de mesas (array de structs Mesa).
- “contadorMesas”: Variável que armazena o número de mesas cadastradas.
- “f_tabela”: Ponteiro para a tabela de funcionários (array de structs Funcionario).
- “contadorFuncionarios”: Variável que armazena o número de funcionários cadastrados.
- “p_tabela”: Ponteiro para a tabela de produtos (array de structs Produto).
- “contadorProdutos”: Variável que armazena o número de produtos cadastrados.

Iremos por partes.

Primeiro pedimos ao utilizador que insira o número da mesa a ser aberta “mesaEscolhida” utilizando “scanf("%d", &mesaEscolhida)”. Pedimos tambem insira o nome do funcionário responsável pela mesa “funcionarioEscolhido” utilizando “scanf("%24s", funcionarioEscolhido)”.

Segundo utilizamos loops “for” para buscar a mesa e o funcionário nas respectivas tabelas, procuramos a mesa com o número inserido “mesaEscolhida” na tabela “m_tabela”. Se a mesa for encontrada define a variável “mesaEncontrada” como “1”. Obtemos um ponteiro para a mesa “mesa”. Se a mesa não for encontrada, “mesaEncontrada” permanece como “0”. Fazemos o mesmo com o funcionario so que procuramos pelo nome inserido “funcionarioEscolhido” na tabela “f_tabela”. Se o funcionário for encontrado definimos a variável “funcionarioEncontrado” como “1”. Se o funcionário não for encontrado, “funcionarioEncontrado” permanece como “0”.

Em terceiro se a mesa e o funcionário forem encontrados “(mesaEncontrada && funcionarioEncontrado)”, verificamos se a mesa já está ocupada “strcmp(mesa->m_disponibilidade, "Ocupada") == 0”. Se a mesa estiver ocupada, é exibida uma mensagem informando que a mesa já está em uso e retorna da função. Se a mesa estiver disponível exibe uma mensagem informando que a mesa foi aberta com sucesso pelo funcionário e define a disponibilidade da mesa como "Ocupada" “strcpy(mesa->m_disponibilidade, "Ocupada")”.

Em quarto para a criação do pedido é alocada memória para um novo pedido “Pedido *novoPedido = (Pedido *)malloc(sizeof(Pedido))”. Inicializamos os campos do novo pedido:

- “n_Mesa”: Define o número da mesa (mesaEscolhida).
- “nomeFuncionario”: Copia o nome do funcionário “strcpy(novoPedido->nomeFuncionario, funcionarioEscolhido)”.
- “produtos”: Inicializa a lista de itens da conta como NULL.
- “contadorProdutos”: Define o contador de itens como 0.
- “total”: Define o total da conta como 0.
- “proximo”: Define o ponteiro para o próximo pedido na lista da mesa “mesa->pedidos”.

Por fim atualizamos a lista de pedidos da mesa para incluir o novo pedido “mesa→pedidos = novoPedido”.

Em quinto Entramos num loop “while” para permitir a adição de itens à conta, para isso pedimos ao utilizador que insira o nome do produto consumido “produtoConsumido” utilizando “scanf("%24s", produtoConsumido)” e a quantidade consumida “quantidadeConsumida” utilizando “scanf("%d", &quantidadeConsumida)”.

Buscamos pelo produto e verificamos a sua quantidade disponível no stock, um loop “for” é usado para percorrer a tabela de produtos “p_tabela”. Comparamos o nome do produto introduzido “produtoConsumido” com o nome de cada produto na tabela. Se o produto for encontrado “strcmp(p_tabela[i].p_nome, produtoConsumido) == 0” e verificamos se a quantidade disponível no estoque é suficiente para a quantidade desejada.

Se a quantidade disponível for suficiente “p_tabela[i].quantidade >= quantidadeConsumida” definimos a variável “produtoEncontrado” como “1”. Armazenamos o preço do produto “preco = p_tabela[i].preco”, atualizamos a quantidade disponível no stock subtraindo a quantidade consumida “p_tabela[i].quantidade -= quantidadeConsumida”.

Se a quantidade disponível for insuficiente “p_tabela[i].quantidade < quantidadeConsumida” Exibe uma mensagem informando que a quantidade do produto é insuficiente.

Se o produto for encontrado “produtoEncontrado”, criamos um novo item de conta. Alocamos memória para um novo item “ItemConta *novoItem = (ItemConta *)malloc(sizeof(ItemConta))”, copiamos o nome do produto para o item “strcpy(novoItem->nomeProduto, produtoConsumido)” e definimos o preço do item “novoItem->preco = preco”. Em seguida designamos a quantidade consumida do item “novoItem-

>quantidade = quantidadeConsumida", atualizamos a lista de itens da conta para incluir o novo item "novoItem->proximo = novoPedido->produtos". Por fim atualizamos o ponteiro para o início da lista de itens da conta "novoPedido->produtos = novoItem".

Em setimo atualizamos o total da conta adicionando o valor do item recém-adicionado "novoPedido->total += preco * quantidadeConsumida" e incrementamos o contador de itens da conta "novoPedido->contadorProdutos++".

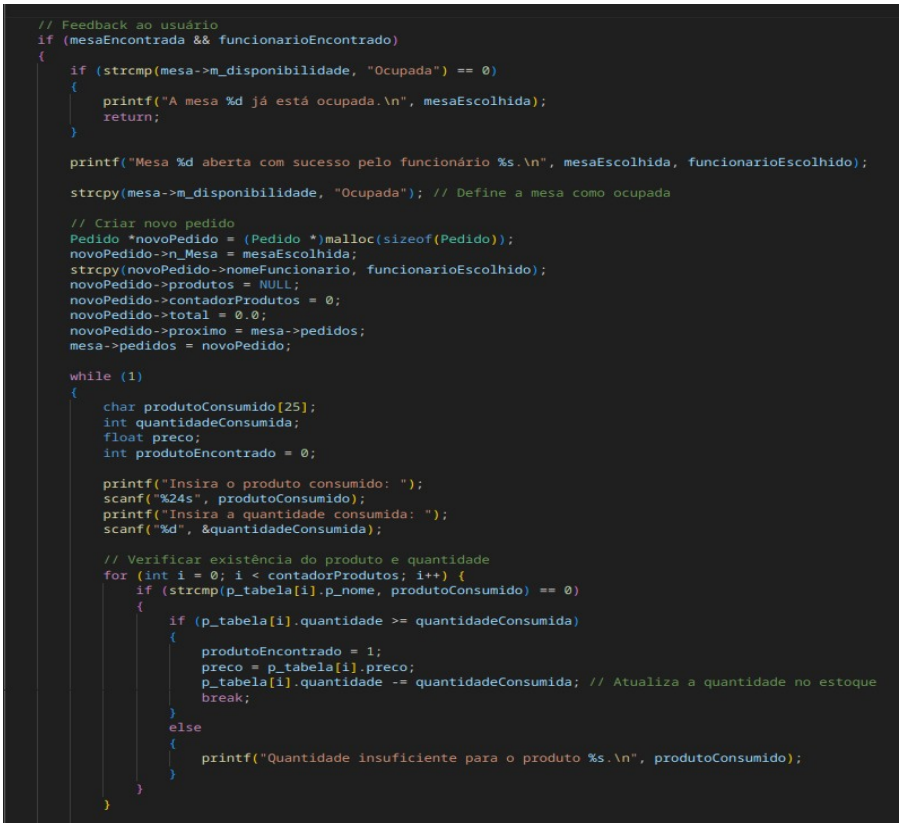
Oitavo perguntamos ao utilizador se deseja adicionar outro produto à conta "printf("Pretende adicionar outro produto? (s/n): ") e "scanf("%2s", resposta)" se a resposta for "s" ou "S", o loop "while" continua. Caso contrário, o loop termina.

Em nono exibimos a conta.

Se a mesa ou o funcionário não forem encontrados "(!mesaEncontrada || !funcionarioEncontrado)", exibimos mensagens informando a situação.

```c
void abrirConta(Mesa *m_tabela, int contadorMesas, Funcionario *f_tabela, int contadorFuncionarios, Produto *p_tabela, int contadorProdutos)
{
    int mesaEscolhida = 0;
    int mesaEncontrada = 0;
    char funcionarioEscolhido[25];
    int funcionarioEncontrado = 0;

    printf("Selecione a mesa a ser aberta: ");
    scanf("%d", &mesaEscolhida);
    printf("Selecione o nome do funcionário responsável: ");
    scanf("%24s", funcionarioEscolhido);

    Mesa *mesa = NULL;

    // Verificar existência da mesa
    for (int i = 0; i < contadorMesas; i++)
    {
        if (m_tabela[i].n_Mesa == mesaEscolhida)
        {
            mesaEncontrada = 1;
            mesa = &m_tabela[i];
            break;
        }
    }

    // Verificar existência do funcionário
    for (int i = 0; i < contadorFuncionarios; i++)
    {
        if (strcmp(f_tabela[i].nome, funcionarioEscolhido) == 0)
        {
            funcionarioEncontrado = 1;
            break;
        }
    }
```

Figure 29: Função para abrir uma conta 1



```c
// Feedback ao usuário
if (mesaEncontrada && funcionarioEncontrado)
{
    if (strcmp(mesa->m_disponibilidade, "Ocupada") == 0)
    {
        printf("A mesa %d já está ocupada.\n", mesaEscolhida);
        return;
    }

    printf("Mesa %d aberta com sucesso pelo funcionário %s.\n", mesaEscolhida, funcionarioEscolhido);

    strcpy(mesa->m_disponibilidade, "Ocupada"); // Define a mesa como ocupada

    // Criar novo pedido
    Pedido *novoPedido = (Pedido *)malloc(sizeof(Pedido));
    novoPedido->n_Mesa = mesaEscolhida;
    strcpy(novoPedido->nomeFuncionario, funcionarioEscolhido);
    novoPedido->produtos = NULL;
    novoPedido->contadorProdutos = 0;
    novoPedido->total = 0.0;
    novoPedido->proximo = mesa->pedidos;
    mesa->pedidos = novoPedido;

    while (1)
    {
        char produtoConsumido[25];
        int quantidadeConsumida;
        float preco;
        int produtoEncontrado = 0;

        printf("Insira o produto consumido: ");
        scanf("%24s", produtoConsumido);
        printf("Insira a quantidade consumida: ");
        scanf("%d", &quantidadeConsumida);

        // Verificar existência do produto e quantidade
        for (int i = 0; i < contadorProdutos; i++) {
            if (strcmp(p_tabela[i].p_nome, produtoConsumido) == 0)
            {
                if (p_tabela[i].quantidade >= quantidadeConsumida)
                {
                    produtoEncontrado = 1;
                    preco = p_tabela[i].preco;
                    p_tabela[i].quantidade -= quantidadeConsumida; // Atualiza a quantidade no estoque
                    break;
                }
                else
                {
                    printf("Quantidade insuficiente para o produto %s.\n", produtoConsumido);
                }
            }
        }
```

Figure 30: Função para inserir uma conta 2

```
        if (produtoEncontrado)
        {
            printf("Produto %s encontrado, preço %.2f.\n", produtoConsumido, preco);

            // Criar novo item de conta
            ItemConta *novoItem = (ItemConta *)malloc(sizeof(ItemConta));
            strcpy(novoItem->nomeProduto, produtoConsumido);
            novoItem->preco = preco;
            novoItem->quantidade = quantidadeConsumida;
            novoItem->proximo = novoPedido->produtos;
            novoPedido->produtos = novoItem;

            novoPedido->total += preco * quantidadeConsumida;
            novoPedido->contadorProdutos++;
        }
        else
        {
            printf("Produto %s não encontrado.\n", produtoConsumido);
        }

        char resposta[3];
        printf("Pretende adicionar outro produto? (s/n): ");
        scanf("%2s", resposta);
        // Limpar o buffer de entrada
        int c;
        while ((c = getchar()) != '\n' && c != EOF) {}

        if (strcmp(resposta, "s") != 0 && strcmp(resposta, "S") != 0)
        {
            break;
        }
    }

    printf("\nConta total para a mesa %d:\n", novoPedido->n_Mesa);
    ItemConta *item = novoPedido->produtos;
    while (item != NULL)
    {
        printf("%s - %d unidade(s) - Preço unitário: %.2f\n", item->nomeProduto, item->quantidade, item->preco);
        item = item->proximo;
    }
    printf("Total: %.2f\n", novoPedido->total);
}
else
{
    if (!mesaEncontrada)
    {
        printf("Mesa %d não existente.\n", mesaEscolhida);
    }
    if (!funcionarioEncontrado)
    {
        printf("Funcionário %s não existente.\n", funcionarioEscolhido);
```

Figure 31: Função para inserir uma conta 3

2.3.6 Atualizar um pedido

Esta função recebe os seguinte parametros:

- “m_tabela”: Ponteiro para a tabela de mesas (array de structs Mesa).
- “contadorMesas”: Variável que armazena o número de mesas cadastradas.
- “f_tabela”: Ponteiro para a tabela de funcionários (array de structs Funcionario).
- “contadorFuncionarios”: Variável que armazena o número de funcionários cadastrados.
- “p_tabela”: Ponteiro para a tabela de produtos (array de structs Produto).
- “contadorProdutos”: Variável que armazena o número de produtos cadastrados.

Assim como todas as outras funções pedimos ao utilizador que insira o número da mesa para a qual deseja atualizar o pedido “mesaEscolhida” utilizando “scanf("%d", &mesaEscolhida)”.

Depois utilizamos um loop for para buscar a mesa na tabela “m_tabela”:

- Se a mesa for encontrada “m_tabela[i].n_Mesa == mesaEscolhida” e definimos a variável “mesaEncontrada” como “1” e tambem obtemos um ponteiro para a mesa “mesa”.

- Se a mesa não for encontrada, “mesaEncontrada” permanece como “0”.

- É exibida uma mensagem informando se a mesa foi encontrada.

Em seguida coletamos o nome do funcionário responsável pelo pedido “funcionarioEscolhido” utilizando “scanf("%24s", funcionarioEscolhido”.

Buscamos pelo funcionario utilizando um loop “for” na tabela “f_tabela”.

Se o funcionário for encontrado “strcmp(f_tabela[i].nome, funcionarioEscolhido) == 0” e definimos a variável “funcionarioEncontrado” como “1” caso contrário “funcionarioEncontrado” permanece como “0”.

E exibimos uma mensagem de feedback.

Logo a seguir verificamos se já existe um pedido aberto para a mesa e o funcionário informados. Para isso percorremos a lista de pedidos da mesa “mesa->pedidos” utilizando um loop “for”, comparando o nome do funcionário de cada pedido “p->nomeFuncionario” com o nome digitado pelo utilizador. Se o funcionário e a mesa coincidirem “strcmp(p->nomeFuncionario, funcionarioEscolhido) == 0” é definido o ponteiro “pedidoExistente” para o pedido encontrado.

Saimos do loop “for” se a combinação de mesa e funcionário não for encontrada e “pedidoExistente” permanece como “NULL”.

Se nenhum pedido for encontrado para a mesa e o funcionário “pedidoExistente == NULL” mostramos uma mensagem informando que um novo pedido será criado.

Criamos um novo pedido “novoPedido” utilizando “malloc()”. Inicializamos os campos do novo pedido:

- “n_Mesa”: Define o número da mesa “mesaEscolhida”.

- “nomeFuncionario”: Copia o nome do funcionário “strcpy(novoPedido->nomeFuncionario, funcionarioEscolhido)”.

- “produtos”: Inicializa a lista de itens da conta como “NULL”.

- “contadorProdutos”: Define o contador de itens como “0”.

- “total”: Define o total da conta como “0”.

- “proximo”: Define o ponteiro para o próximo pedido na lista da mesa “mesa->pedidos”.

Por fim atualizamos a lista de pedidos da mesa para incluir o novo pedido “mesa->pedidos = novoPedido”e definimos “pedidoExistente” como o novo pedido “pedidoExistente = novoPedido”. Entramos em um loop “while” para permitir a adição ou modificação de itens na conta.

A seguir pedimos ao utilizador que insira a quantidade do produto consumido “quantidadeConsumida” utilizando “scanf("%d", &quantidadeConsumida)”.

Utilizamos um loop “for” para buscar o produto na tabela “p_tabela” para “comparar” o nome do produto inserido “produtoConsumido” com o nome de cada produto na tabela.

Se o produto for encontrado “strcmp(p_tabela[i].p_nome, produtoConsumido) == 0”:

- Verificamos se a quantidade disponível no estoque é suficiente para a quantidade desejada “p_tabela[i].quantidade >= quantidadeConsumida”
- Se a quantidade for suficiente, definimos a variável “produtoEncontrado” como “1”.
- Armazenamos o preço do produto “preco = p_tabela[i].preco”.
- E atualizamos a quantidade disponível no estoque subtraindo a quantidade consumida “p_tabela[i].quantidade -= quantidadeConsumida”.

Se a quantidade for insuficiente ”p_tabela[i].quantidade < quantidadeConsumida” exibimos uma mesnagem a informar que a quantidae do produto é insuficiente

No proximo passo criamos o novo Item de Conta (se necessário).

Se o produto for encontrado “produtoEncontrado” verificamos se o produto já está na lista de itens do pedido “pedidoExistente->produtos” para isso percorremos a lista de itens do pedido utilizando um loop “for” para comparar o nome do produto de cada item “item->nomeProduto” com o nome do produto encontrado. Se o produto já estiver na lista “strcmp(item->nomeProduto, produtoConsumido) == 0” e definimos o ponteiro “itemExistente” para o item da lista e saimos do loop “for”.

Caso o produto não estiver na lista “itemExistente == NULL” criamos um novo item de conta “novoItem” utilizando “malloc()”, inicializando os campos do novo item:

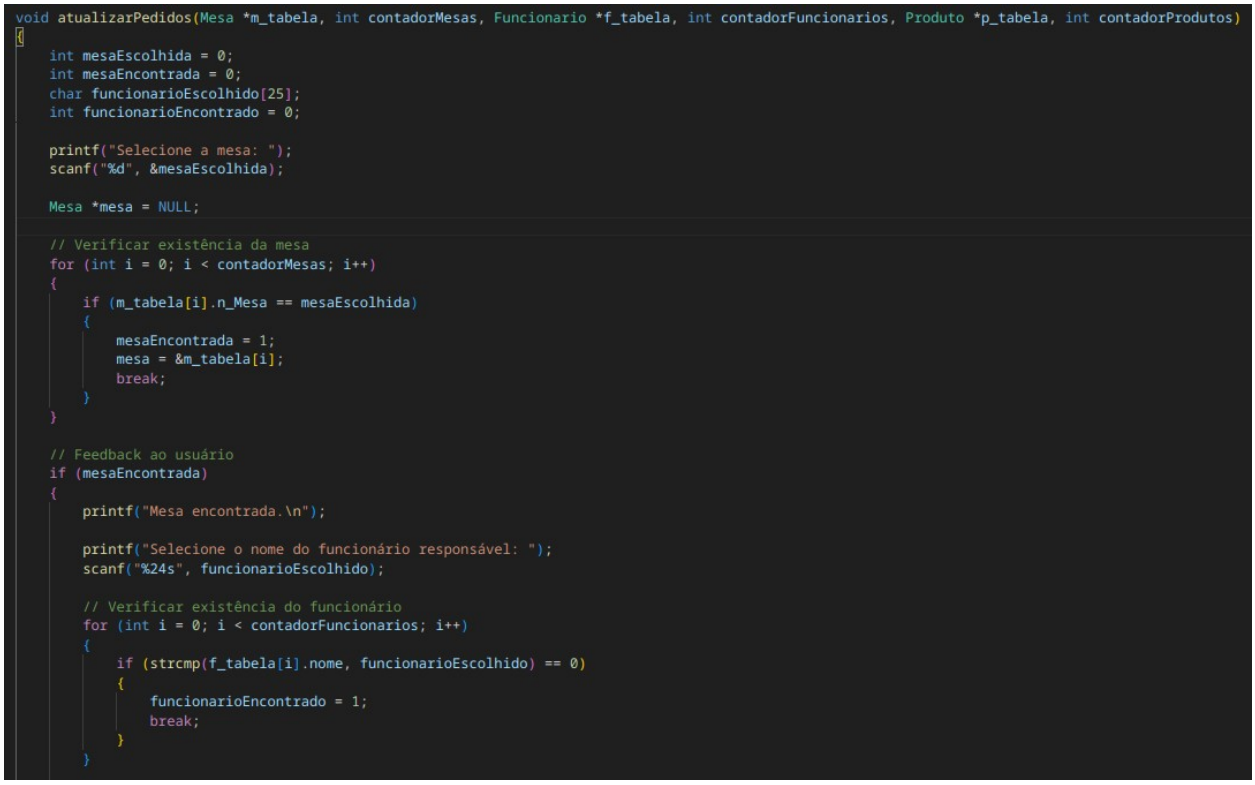
- “nomeProduto: Copia o nome do produto “strcpy(novoItem->nomeProduto, produtoConsumido)”;
- “preco: Define o preço do produto “novoItem->preco = preco”;
- “quantidade”: Define a quantidade consumida do produto “novoItem->quantidade = quantidadeConsumida”;
- “proximo”: Define o ponteiro para o próximo item na lista da conta “novoItem->proximo = pedidoExistente->produtos”;

Atualizamos a lista de itens da conta para incluir o novo item “pedidoExistente->produtos = novoItem” e o total da conta adicionando o valor do item recém-adicionado "pedidoExistente->total += preco * quantidadeConsumida”, por fim não pode faltar a incrementação do contador de itens da conta “pedidoExistente->contadorProdutos++”.

Se necessário tambem podemos atualizar o item existente ou seja:

Se o produto já estiver na lista de itens “itemExistente != NULL” atualizamos a quantidade do item na lista “itemExistente->quantidade += quantidadeConsumida” e o total da conta “pedidoExistente->total += preco * quantidadeConsumida;”.

Por fim perguntamos ao utilizador se pretende atualizar ou modificar outro item na conta, caso positivo continua o ciclo, caso negativo é paresentada a conta.



```
void atualizarPedidos(Mesa *m_tabela, int contadorMesas, Funcionario *f_tabela, int contadorFuncionarios, Produto *p_tabela, int contadorProdutos)
{
    int mesaEscolhida = 0;
    int mesaEncontrada = 0;
    char funcionarioEscolhido[25];
    int funcionarioEncontrado = 0;

    printf("Selecione a mesa: ");
    scanf("%d", &mesaEscolhida);

    Mesa *mesa = NULL;

    // Verificar existência da mesa
    for (int i = 0; i < contadorMesas; i++)
    {
        if (m_tabela[i].n_Mesa == mesaEscolhida)
        {
            mesaEncontrada = 1;
            mesa = &m_tabela[i];
            break;
        }
    }

    // Feedback ao usuário
    if (mesaEncontrada)
    {
        printf("Mesa encontrada.\n");

        printf("Selecione o nome do funcionário responsável: ");
        scanf("%24s", funcionarioEscolhido);

        // Verificar existência do funcionário
        for (int i = 0; i < contadorFuncionarios; i++)
        {
            if (strcmp(f_tabela[i].nome, funcionarioEscolhido) == 0)
            {
                funcionarioEncontrado = 1;
                break;
            }
        }
```

Figure 32: Função para atualizar o pedido 1

```c
if (funcionarioEncontrado)
{
    printf("Funcionário encontrado.\n");

    // Verificar se já existe um pedido para esta mesa
    Pedido *pedidoExistente = NULL;
    for (Pedido *p = mesa->pedidos; p != NULL; p = p->proximo)
    {
        if (strcmp(p->nomeFuncionario, funcionarioEscolhido) == 0)
        {
            pedidoExistente = p;
            break;
        }
    }

    if (pedidoExistente == NULL)
    {
        printf("Criando novo pedido.\n");

        // Criar novo pedido
        Pedido *novoPedido = (Pedido *)malloc(sizeof(Pedido));
        novoPedido->n_Mesa = mesaEscolhida;
        strcpy(novoPedido->nomeFuncionario, funcionarioEscolhido);
        novoPedido->produtos = NULL;
        novoPedido->contadorProdutos = 0;
        novoPedido->total = 0.0;
        novoPedido->proximo = mesa->pedidos;
        mesa->pedidos = novoPedido;
        pedidoExistente = novoPedido;
    }

    while (1)
    {
        char produtoConsumido[25];
        int quantidadeConsumida;
        float preco;
        int produtoEncontrado = 0;

        printf("Insira o produto consumido: ");
        scanf("%24s", produtoConsumido);
        printf("Insira a quantidade consumida: ");
        scanf("%d", &quantidadeConsumida);
```

Figure 33: Função para atualizar um pedido 2

```c
// Verificar existência do produto e quantidade
for (int i = 0; i < contadorProdutos; i++)
{
    if (strcmp(p_tabela[i].p_nome, produtoConsumido) == 0)
    {
        if (p_tabela[i].quantidade >= quantidadeConsumida)
        {
            produtoEncontrado = 1;
            preco = p_tabela[i].preco;
            p_tabela[i].quantidade -= quantidadeConsumida; // Atualiza a quantidade no estoque
            break;
        }
        else
        {
            printf("Quantidade insuficiente para o produto %s.\n", produtoConsumido);
        }
    }
}

if (produtoEncontrado)
{
    printf("Produto %s encontrado, preço %.2f.\n", produtoConsumido, preco);

    // Verificar se o produto já está na lista de itens do pedido
    ItemConta *itemExistente = NULL;
    for (ItemConta *item = pedidoExistente->produtos; item != NULL; item = item->proximo)
    {
        if (strcmp(item->nomeProduto, produtoConsumido) == 0)
        {
            itemExistente = item;
            break;
        }
    }

    if (itemExistente == NULL)
    {
        // Criar novo item de conta
        ItemConta *novoItem = (ItemConta *)malloc(sizeof(ItemConta));
        strcpy(novoItem->nomeProduto, produtoConsumido);
        novoItem->preco = preco;
        novoItem->quantidade = quantidadeConsumida;
        novoItem->proximo = pedidoExistente->produtos;
        pedidoExistente->produtos = novoItem;
        pedidoExistente->total += preco * quantidadeConsumida;
        pedidoExistente->contadorProdutos++;
    }
    else
    {
        // Produto já existe no pedido, apenas atualize a quantidade e o total
        itemExistente->quantidade += quantidadeConsumida;
        pedidoExistente->total += preco * quantidadeConsumida;
    }
```

```c
            else
            {
                printf("Produto %s não encontrado.\n", produtoConsumido);
            }

            char resposta[3];
            printf("Pretende adicionar outro produto? (s/n): ");
            scanf("%2s", resposta);
            // Limpar o buffer de entrada
            int c;
            while ((c = getchar()) != '\n' && c != EOF) {}

            if (strcmp(resposta, "s") != 0 && strcmp(resposta, "S") != 0)
            {
                break;
            }
        }

        printf("\nConta total para a mesa %d:\n", pedidoExistente->n_Mesa);
        ItemConta *item = pedidoExistente->produtos;
        while (item != NULL)
        {
            printf("%s - %d unidade(s) - Preço unitário: %.2f\n", item->nomeProduto, item->quantidade, item->preco);
            item = item->proximo;
        }
        printf("Total: %.2f\n", pedidoExistente->total);
    }
    else
    {
        printf("Funcionário %s não encontrado.\n", funcionarioEscolhido);
    }
}
else
{
    printf("Mesa %d não encontrada.\n", mesaEscolhida);
}

// Limpar o buffer de entrada
int c;
while ((c = getchar()) != '\n' && c != EOF) {}

printf("\nPressione Enter para continuar...\n");
getchar(); // Espera que o utilizador pressione Enter
}
```

Figure 34: Função para atualizar um pedido 4

2.3.7 Pagar um pedido

Pede ao utlizador que insira o número da mesa para a qual deseja pagar o pedido “mesaEscolhida” utilizando “scanf("%d", &mesaEscolhida)”.

Em seguida usamos um loop “for” para procurar a mesa na tabela “m_tabela” percorremos cada mesa na tabela “i < *contador” e “compara” o número da mesa digitado “mesaEscolhida” com o número da mesa atual “m_tabela[i].n_Mesa”.

Se a mesa for encontrada “m_tabela[i].n_Mesa == mesaEscolhida” e definimos a variável “mesaEncontrada” como “1” e tambem obtemos um ponteiro para a mesa “mesa = &m_tabela[i]”.

Saimos do loop for. Se a mesa não for encontrada por consequente “mesaEncontrada” permanece como “0”.

No pagamento definimos a disponibilidade da mesa como "livre" “strcpy(mesa->m_disponibilidade, "livre")”. Removemos o pedido associado à mesa e armazena o ponteiro para o pedido atual “Pedido *pedidoAtual = mesa->pedidos”.

Atualizamos o ponteiro para o próximo pedido na lista da mesa “mesa->pedidos = pedidoAtual->proximo”. Esta ação libera a memória do pedido atual e de seus itens, finalizando a conta e exibe uma mensagem informando que o pagamento foi realizado com sucesso.

```c
void pagarPedido(Mesa *m_tabela, int *contador)
{
    int mesaEscolhida;
    int mesaEncontrada = 0;

    printf("Selecione a mesa para pagar o pedido: ");
    scanf("%d", &mesaEscolhida);

    // Encontrar a mesa correspondente na lista
    for (int i = 0; i < *contador; i++)
    {
        if (m_tabela[i].n_Mesa == mesaEscolhida)
        {
            mesaEncontrada = 1;
            Mesa *mesa = &m_tabela[i];

            // Define a mesa como livre após o pagamento
            strcpy(mesa->m_disponibilidade, "livre");

            // Remover o pedido associado à mesa
            Pedido *pedidoAtual = mesa->pedidos;
            mesa->pedidos = pedidoAtual->proximo; // Avançar para o próximo pedido
            break;
        }

    }

    if (mesaEncontrada)
    {
        printf("Mesa %d encontrada.\n", mesaEscolhida);
        printf("O pagamento foi realizado com sucesso.\n");
    }
    else
    {
        printf("Mesa %d não encontrada.\n", mesaEscolhida);
    }
}
```

Figure 35: Função para pagar um pedido

2.3.8 Menu das mesas

A semelhança dos outros menus este menu é só mais um ciclo “while” com “switch case”.

```c
void menuMesas(Mesa *m_tabela, int *contador, Funcionario *f_tabela, int *contadorFuncionarios, Produto *p_tabela, int *contadorProdutos)
{
    int opcao;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Adicionar Mesa\n");
        printf("2. Ver Mesas\n");
        printf("3. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        // Limpar o buffer de entrada
        int c;
        while ((c = getchar()) != '\n' && c != EOF) { }

        switch (opcao)
        {
            case 1:
                system("clear");
                adicionarMesa(m_tabela, contador);
                break;
            case 2:
                system("clear");
                mostrarMesas(m_tabela, *contador);
                break;
            case 3:
                printf("Saindo...\n");
                return;
            default:
                printf("Opção inválida. Tente novamente.\n");
        }
    }
}
```

Figure 36: Menu das mesas

2.4 Encomendas

A função das encomendas recebe os seguintes parametros.

- “e_tabela”: Ponteiro para a tabela de encomendas “array de structs Encomenda”.
- “E_contador”: Ponteiro para a variável que armazena o número de encomendas registadas.
- “P_tabela”: Ponteiro para a tabela de produtos “array de structs Produto”.
- “P_contador”: Variável que armazena o número de produtos registados.

Verificamos se o número de encomendas registadas “*E_contador” já atingiu o limite máximo “MAX_ENCOMENDAS”. Se o limite for atingido, exibe uma mensagem informando ao usuário e retorna da função.

Obtemos a encomenda atual através um ponteiro para a próxima encomenda disponível na tabela “Encomenda *encomenda = &e_tabela[*E_contador]”.

Pedimos ao utilizador que insira o nome do produto a ser encomendado #printf("Insira o produto a encomendar: ")” e utilizando “fgets(encomenda->nome, sizeof(encomenda->nome), stdin)”.

A função fgets lê uma linha do teclado e a armazena no campo nome da struct encomenda, incluindo espaços em branco e a quebra de linha (\n). A função “strcspn()” é utilizada para remover a quebra de linha do final da string “encomenda->nome[strcspn(encomenda->nome, "\n")] = '\0'”.

Pedimos ao utilizador que insira tambem a quantidade do produto desejada “printf("Insira a quantidade a encomendar: ")” utilizando “scanf("%d", &(encomenda->quantidade)”.

Utilizamos um loop for para realizar a busca do produto na tabela “p_tabela”, percorremos cada produto na tabela “i < P_contador”.

Comparamos o nome do produto inserido “encomenda->nome” com o nome do produto atual “p_tabela[i].p_nome”. Se o produto for encontrado “strcmp(p_tabela[i].p_nome, encomenda->nome) == 0”

Verifica se a quantidade disponível no estoque “p_tabela[i].quantidade” é suficiente para atender à quantidade desejada “encomenda->quantidade”. Se a quantidade for suficiente “p_tabela[i].quantidade >= encomenda->quantidade”.

Atualizamos a quantidade disponível no estoque subtraindo a quantidade encomendada “p_tabela[i].quantidade -= encomenda->quantidade” e define a variável produtoEncontrado como “1” para indicar que o produto foi encontrado e a quantidade foi atualizada.

Se a quantidade for insuficiente “p_tabela[i].quantidade < encomenda->quantidade” e exibe uma mensagem informando que a quantidade do produto é insuficiente e retorna da função, Se o produto não for encontrado “!produtoEncontrado”, a variável produtoEncontrado permanece como “0”.

```c
void encomendas(Encomenda *e_tabela, int *E_contador, Produto *p_tabela, int P_contador)
{
    if (*E_contador >= MAX_ENCOMENDAS)
    {
        printf("Limite de encomendas atingido!\n");
        return;
    }

    Encomenda *encomenda = &e_tabela[*E_contador];

    // Limpar o buffer do teclado
    int c;
    while ((c = getchar()) != '\n' && c != EOF) { }

    printf("Insira o produto a encomendar: ");
    fgets(encomenda->nome, sizeof(encomenda->nome), stdin);
    encomenda->nome[strcspn(encomenda->nome, "\n")] = '\0'; // Remover a quebra de linha do fgets

    printf("Insira a quantidade a encomendar: ");
    scanf("%d", &(encomenda->quantidade));

    // Limpar o buffer do teclado
    while ((c = getchar()) != '\n' && c != EOF) { }

    // Verificar se o produto existe e atualizar a quantidade disponível
    int produtoEncontrado = 0;
    for (int i = 0; i < P_contador; i++)
    {
        if (strcmp(p_tabela[i].p_nome, encomenda->nome) == 0)
        {
            if (p_tabela[i].quantidade >= encomenda->quantidade)
            {
                p_tabela[i].quantidade -= encomenda->quantidade;
                produtoEncontrado = 1;
            }
            else
            {
                printf("Quantidade insuficiente do produto '%s'.\n", encomenda->nome);
                return;
            }
        }
    }

    if (!produtoEncontrado)
    {
        printf("Produto '%s' não registrado no sistema.\n", encomenda->nome);
        return;
    }
```

Figure 37: Função para realizar encomenda 1

```
    system("clear");
    printf("\nPressione Enter para mostrar os dados da encomenda inserida...\n");
    getchar(); // Espera que o utilizador pressione Enter

    printf("\n------------------Encomenda adicionada:------------------\n");
    printf("Nome: %s\n", encomenda->nome);
    printf("Quantidade: %d\n", encomenda->quantidade);
    printf("\n---------------------------------------------------------------\n");

    (*E_contador)++;

    printf("\nPressione Enter para continuar...\n");
    while ((c = getchar()) != '\n' && c != EOF) { }
    getchar(); // Espera que o utilizador pressione Enter
}
```

Figure 38: Função para realizar encomendas 2

2.5 Menu Principal

A função menuPrincipal é um menu que permite fazer o registo de tudo.

```
void menuPrincipal(Mesa *m_tabela, int *M_contador, Funcionario *f_tabela, int *F_contador, Produto *p_tabela, int *P_contador, Encomenda *e_tabela, int *E_contador)
{
    int opcao;
    escreverPedidosCSV(m_tabela, *M_contador, e_tabela, *E_contador, p_tabela, P_contador);

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Funcionários\n");
        printf("2. Produtos\n");
        printf("3. Mesas\n");
        printf("4. Encomendas\n");
        printf("5. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        // Limpar o buffer de entrada
        int c;
        while ((c = getchar()) != '\n' && c != EOF) {}

        switch (opcao)
        {
            case 1:
                system("clear");
                menuFuncionarios(f_tabela, F_contador);
                break;
            case 2:
                system("clear");
                menuProdutos(p_tabela, P_contador);
                break;
            case 3:
                system("clear");
                menuMesas(m_tabela, M_contador, f_tabela, F_contador, p_tabela, P_contador);
                break;
            case 4:
                system("clear");
                encomendas(e_tabela, E_contador, p_tabela, P_contador);
                break;
            case 5:
                printf("Saindo...\n");
                return;
            default:
                printf("Opção inválida. Tente novamente.\n");
        }

    }
}
```

Figure 39: Menu Principal

2.6 Menu Principal 2

Função menuPrincipal2 permite realizar o registo dos pedidos e ver as mesas

```c
void menuPrincipal2(Mesa *m_tabela, int *M_contador, Funcionario *f_tabela, int *F_contador, Produto *p_tabela, int *P_contador)
{
    int opcao;

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Realizar Pedido\n");
        printf("2. atualizar um pedido\n");
        printf("3. Pagar Pedido\n");
        printf("4. Ver Mesas\n");
        printf("5. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        // Limpar o buffer de entrada
        int c;
        while ((c = getchar()) != '\n' && c != EOF) {}

        switch (opcao)
        {
            case 1:
               system("clear");
               abrirConta(m_tabela, *M_contador, f_tabela, *F_contador, p_tabela, *P_contador);
                break;
            case 2:
                system("clear");
                atualizarPedidos(m_tabela, *M_contador, f_tabela, *F_contador, p_tabela, *P_contador);
                break;
            case 3:
                system("clear");
                pagarPedido(m_tabela, M_contador);
                break;
            case 5:
                printf("Saindo...\n");
                return;
            default:
                printf("Opção inválida. Tente novamente.\n");
        }
    }
}
```

Funcionario *f_tabela

Figure 40: Menu principal 2

2.7 Função main

Função que leva para os restantes menus e inicia todas as tabelas e todos os contadores, tambem é responsavel por carregar a informação dos ficheiro binarios e a ultima opção escreve os pedidos em ficheiros .csv.

```c
int main()
{
    int opcao;
    Encomenda e_tabela[MAX_ENCOMENDAS];
    Funcionario f_tabela[MAX_FUNCIONARIOS];
    Produto p_tabela[MAX_PRODUTOS];
    Mesa m_tabela[MAX_MESAS];
    int E_contador = 0;
    int M_contador = 0;
    int P_contador = 0;
    int F_contador = 0;

    // Carregar os dados dos arquivos binários
    P_contador = carregarProdutos(p_tabela);
    F_contador = carregarFuncionarios(f_tabela);
    M_contador = carregarMesas(m_tabela);

    while (1)
    {
        printf("\nMenu:\n");
        printf("1. Abrir dia\n");
        printf("2. Fechar dia\n");
        printf("3. Sair\n");
        printf("Escolha uma opção: ");
        scanf("%d", &opcao);

        // Limpar o buffer de entrada
        int c;
        while ((c = getchar()) != '\n' && c != EOF) {}

        switch (opcao)
        {
            case 1:
                system("clear");
                // Supondo que menuPrincipal2 existe e foi definido anteriormente
                menuPrincipal2(m_tabela, &M_contador, f_tabela, &F_contador, p_tabela, &P_contador);
                break;
            case 2:
                system("clear");
                // Supondo que menuPrincipal existe e foi definido anteriormente
                menuPrincipal(m_tabela, &M_contador, f_tabela, &F_contador, p_tabela, &P_contador, e_tabela, &E_contador);
                break;
            case 3:
                printf("Saindo...\n");
                // Salvar os dados nos arquivos binários
                salvarProdutos(p_tabela, P_contador);
                salvarFuncionarios(f_tabela, F_contador);
                salvarMesas(m_tabela, M_contador);
                // Escrever pedidos em CSV
                escreverPedidosCSV(m_tabela, M_contador, e_tabela, E_contador, p_tabela, &P_contador);
                return 0;
            default:
```

Figure 41: Função main

3 Conclusão

Com este trabalho foi possivel aplicar todo o conhecimento de “C” e aplicação avançada de listas e organuzação das mesmas, apesar de nao haver certeza de que foram aplicadas corretamente.