ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Projeto de Laboratório de Programação

Licenciatura em Engenharia Informática

2022/2023

Grupo 107

César Castelo – 8220169

Hugo Guimarães – 8220337

Pedro Pinho – 8220307

Índice

Índice de Figuras

Chave de siglas

CRUD	Create, Read, Update, Delete
CSV	Character Separated Values

1. Introdução

No âmbito da unidade curricular de Laboratório de Programação, foi realizado um projeto que funcionará como elemento integrador dos conhecimentos adquiridos no decorrer das aulas de laboratório de programação e de fundamentos de programação.

Este projeto, tem como premissa inicial, um programa escrito na linguagem de programação C, que deverá permitir registar, alterar e eliminar dados referentes a diversos campos, sendo que, esses dados têm que ser guardados em ficheiros separados, para que os mesmo se mantenham mesmo após o encerramento do programa.

2. Funcionalidades requeridas

Neste projeto foi pedido que se realizasse um programa que permite iniciar sessão como cliente ou administrador. Sendo que cada um desses perfis tem diferentes funcionalidades, sendo elas:

Administrador:

- Gestão de Clientes;
- Gestão de Produtos;
- Gestão de Materiais;
- Gestão de Produção.

Clientes:

- Registo de uma encomenda.

Juntamente com estas funcionalidades, também foi pedido para que os dados fossem guardados em ficheiros separados, para que eles não se perdessem quando o programa fosse encerrado. Sendo que os dados devem ser guardados nos respetivos ficheiros, somente quando o utilizador desejar guardar as operações que realizou, a partir de uma opção no respetivo menu.

Para além de ser pedido para que usássemos ficheiros externos, também nos foi pedido para que fosse usada memória dinâmica sempre que achássemos necessário, para que não houvesse limitações no armazenamento dos dados nas variáveis.

Estruturas

Para que os dados fossem organizados da melhor forma possível, foram criadas várias estruturas para que fosse mais fácil adicionar, remover e atualizar os dados da forma mais prática e organizada possível.

As estruturas estão organizadas da seguinte forma:

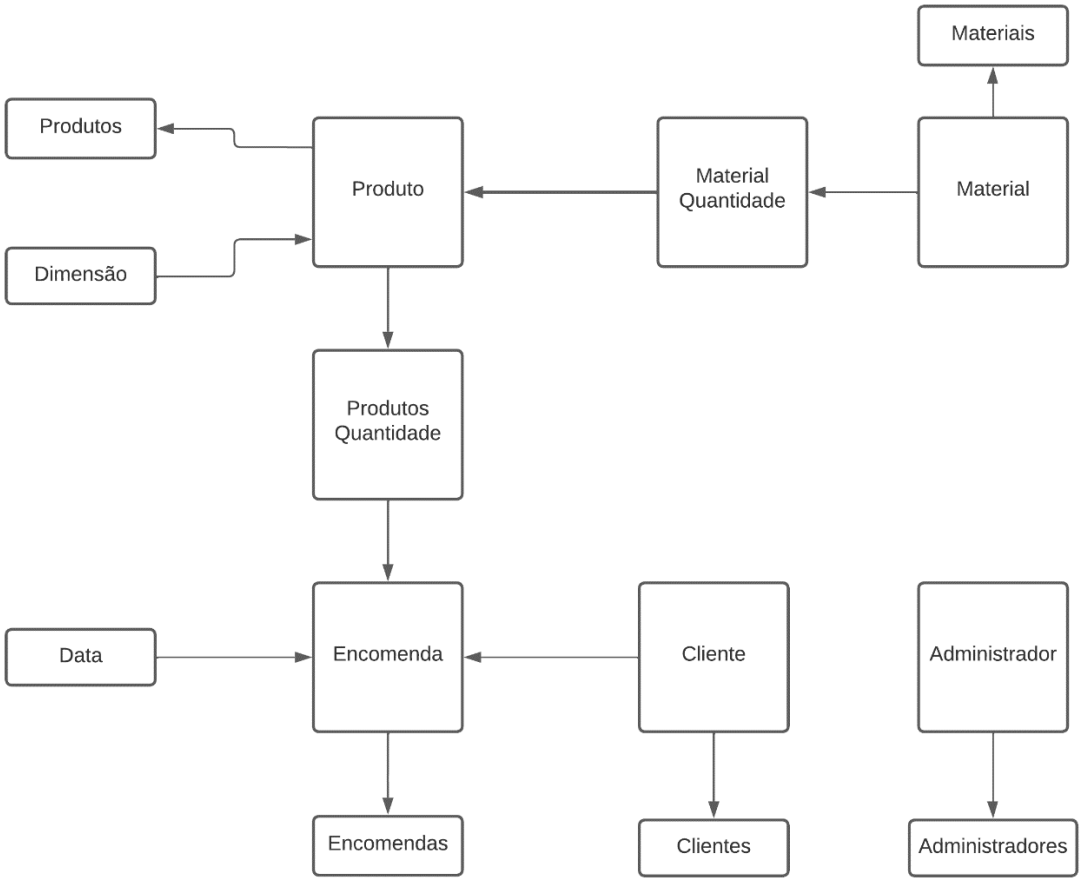

Figura 1 - Diagrama de estruturas

a) Clientes

```
/** @struct Cliente
 *  @brief @brief Estrutura que guarda a informação de um cliente.
 */
typedef struct {
    char codCliente[INDICE_COD_CLIENTE];
    char nome[INDICE_NOME_CLIENTE];
    char morada[INDICE_MORADA_CLIENTE];
    int nif;
    char origem[INDICE_ORIGEM_CLIENTE];
    char password[INDICE_PASSWORD_CLIENTE];
    int quantidadeEncomendas;
    Estado estado;
} Cliente;

/** @struct Clientes
 *  @brief Estrutra que guarda a informação de todos os clientes.
 */
typedef struct {
    int contador;
    Cliente *cliente;
    int tamanhoArray;
} Clientes;
```

Figura 2 - Estruturas dos Clientes

A estrutura Clientes, contém tudo aquilo que pode ser necessário armazenar de um cliente, mas, o que se destaca mais é a variável estado, que é basicamente um enum[1], que só pode receber os valores 0(INATIVO) e 1 (ATIVO). Quando um utilizador é criado, por defeito vem como inativo, pois ainda não realizou nenhuma encomenda, mas a partir do momento que esta for realizada, o seu estado fica ativo, até à data da entrega da encomenda.

Já a estrutura Clientes, serve para armazenar todos os clientes que foram inseridos no programa, sendo a variável contador, o número que identifica a posição de cada cliente no array[2], e a variável, tamanhoArray é o valor que será posto dentro do malloc [3]que será executado quando o programa for criado, ele é a soma da variável contador com uma margem de vinte valores. Isto serve para que não se precise realocar mais memoria sempre que o utilizador insira um novo utilizador, basta apenas faze-lo de vinte em vinte adições aos clientes, assim sendo mantendo um bom desempenho do programa, caso haja muitos Clientes já armazenados

O cliente para poder entrar e poder realizar uma encomenda, precisa de colocar o seu código de cliente e a sua password

[1] é um conjunto de valores inteiros representados por identificadores.
[2] é um conjunto de endereços na memória RAM que são relacionados entre si por possuírem o mesmo nome e tipo de dados.
[3] aloca memória na heap de acordo com o tamanho passado, e retorna um apontador para o local onde houve a alocação

b) Materiais

```c
/** @struct Material
 *  @brief Estrutura que guarda a informação de um Material.
 */
typedef struct {
    char codMatrial[INDICE_COD_MATERIAL];
    char descricao[INDICE_DESCRICAO_MATERIAL];
    char unidade[INDICE_UNIDADE_MATERIAL];
    int vezesUsadas; ///< quantidade de vezes que um material foi preciso para fazer os produtos
} Material;

/** @struct Materiais
 *  @brief Estrutra que guarda a informação de todos as Materiais.
 */
typedef struct {
    int contador;
    Material *material;
    int tamanhoArray;
} Materiais;
```

Figura 3 - Estruturas dos Materiais

As estruturas dos materiais são parecidas com as estruturas dos clientes e dos administradores, mudando apenas o nome das estruturas, e aquilo que é guardado nelas, mais especificamente na estrutura material, que contêm os relevantes para a inserção de um material. Sendo eles o código do material, a descrição, a quantidade de vezes usadas, e a unidade do mesmo, este último corresponde à unidade do mesmo material, como por exemplo, se o material for sólido, então a unidade do mesmo será em gramas ou quilogramas, mas se o material for líquido, então a unidade do material será em litros ou mililitros.

c) Produtos

```c
/** @struct Material_Quantidade
 *  @brief Estrutura que guarda quantidade de do material inserido para a confeccao do produto.
 */
typedef struct {
    char codMatrial[INDICE_COD_MATERIAL];
    int quantidade;
} Material_Quantidade;

/** @struct Produto
 *  @brief Estrutura que guarda a informação de um Produto.
 */
typedef struct {
    char codProduto[INDICE_COD_PRODUTO];
    char nome[50];
    Dimensao dimensoes;
    int quantidadeMateriais; ///< Quantidade de materiais diferentes sao necessarios para fabricar o produto
    Material_Quantidade *materiais; ///< Guarda o codigo do produto e a quantidade necessaria desse material
    Estado estado; ///< 0 - poduto desativado, 1 - produto ativo (que ja foi requesitado numa encomenda)
    float preco; ///< Sera pedido ao utilizador o preco bruto do produto, mas o que sera guardado sera o preco liquido.
    Bool estaRemovido; ///< indica se um produto que ja foi requesitado foi removido ou nao, 0 - false, 1 - true
    int vezesUsadas; ///< quantidade de vezes que um produto foi requesitado em encomendas
} Produto;

/** @struct Produtos
 *  @brief Estrutra que guarda a informação de todos os produtos.
 */
typedef struct {
    int contador; ///< numero de produtos existentes no array
    Produto *produto;
    int tamanhoArray; ///< tamanho do array que sera criado para os produtos, que tem sempre mais do que aquilo que guarda
} Produtos;
```

Figura 4 - Estruturas dos Produtos

Nos produtos, ao contrário das outras estruturas, tem uma estrutura a mais, que especifica a quantidade de material que é necessário para fazer esse produto em específico, é por isso que a variável materiais do tipo “Material_Quantidade” que fica na estrutura “Produto” é um apontador, pois para fazer um determinado produto, são precisos múltiplos materiais, que precisarão ser armazenados em um array, e como não se sabe a quantidade de materiais, é preciso alocar e realocar a memoria manualmente, assim o administrador pode introduzir todos os materiais necessários sem restrições de quantidade.

Os campos na estrutura “Produto” são organizados de acordo com a informação que é relevante para um produto, que são: o código do produto; o nome; as dimensões, que aqui estão a ser guardadas em uma variável do tipo “Dimensoes”, que é uma estrutura que foi criada para facilitar a inserção, e organização dos dados; a quantidade de materiais necessários para fazer o produto em questão; os materiais, juntamente com a quantidade dos mesmos, que, como foi dito anteriormente, vão ser guardados na variável do tipo “Material_Quantidade”; o estado do produto, 0(INATIVO) ou 1(ATIVO), a quantidade de vezes usadas, e o preço do mesmo.

O estado do produto serve para indicar se esse produto já foi inserido em alguma encomenda ou não, pois caso o produto já tenha sido pedido em uma encomenda, ele não pode ser removido.

d) Encomendas

```c
/** @struct Produto_Quantidade
 *  @brief Estrutura que guarda quantidade que o cliente quer do produto que inseriu na encomenda.
 */
typedef struct {
    char codProduto[INDICE_COD_PRODUTO];
    int quantidade;
} Produto_Quantidade;

/** @struct Encomenda
 *  @brief Estrutura que guarda a informação de uma Encomenda.
 */
typedef struct {
    char codEncomenda[INDICE_COD_ENCOMENDA];
    char codCliente[INDICE_COD_CLIENTE];
    int quantidadeProdutos;
    Produto_Quantidade *produtos;
    Data dataRequerimento; ///< Indica o dia em que foi requesiada a encomenda
    Data dataEntrega;
    Estado estado; ///< Se ja passou o prazo de entrega ou nao
    float preco;
} Encomenda;

/** @struct Encomendas
 *  @brief Estrutra que guarda a informação de todos as encomendas.
 */
typedef struct {
    int contador;
    Encomenda *encomenda;
    int tamanhoArray;
} Encomendas;
```

Figura 5 - Estrutura das Encomendas

As estruturas das encomendas são muito parecidas com as estruturas dos produtos, mas no caso dos produtos, para construir um produto, são necessários múltiplos materiais em quantidades diferentes, aqui, para realizar uma encomenda, podem ser precisos múltiplos materiais em quantidades diferentes.

Nas encomendas, foram colocados os seguintes campos: Código da encomenda; Código de cliente que realizou a encomenda; quantidade de produtos que o cliente requereu; os produtos e a quantidade que o cliente inseriu; a data de requerimento que indica o dia em que foi inserida a encomenda, que é do tipo Data, que é uma estrutura criada para facilitar a inserção e organização das datas; a data da entrega da encomenda, que também é do tipo data; os estado da encomenda, 0 (INATIVO) e 1 (ATIVO), que indica se já passou do prazo de entrega ou não; e o preço total da encomenda, que é soma dos preços dos produtos.

3. Funcionalidades propostas

Listagens distintas

Para além das listagens de cada estrutura e das propostas para a produção, também foram propostas cinco listagens distintas que devem ser do interesse da empresa. O principal objetivo dessas listagens é avaliar a compreensão do problema, e a capacidade de analisar os dados armazenados.

• Países de origem mais frequentes

Uma empresa, embora venda para múltiplos países, é normal que apareçam mais clientes de uns países do que de outros, logo, acaba por ser crucial para a empresa, saber onde o seu público alvo mais se concentra. Logo no programa foi implementada uma função que lista e ordena por ordem decrescente os países de origem mais frequentes entre os clientes da empresa.

Esta função cria e preenche uma lista com os países de origem dos clientes, e a quantidade de clientes registados de cada país. Ela aloca memoria para a estrutura de dados "Listagens" e, a seguir, percorre a lista de clientes para contar quantos clientes de cada país existem. Se um país já estiver na lista, o contador correspondente é incrementado. Caso não exista, o país é adicionado à lista e o contador é definido como 1.

A função também usa outra função chamada "ordenarListagens" para classificar a lista de países por ordem decrescente em quantidade de clientes, e uma função chamada "printArrayListagens" para imprimir os resultados. Por fim, liberta a memoria alocada.

```
void listagemTopPaisesMaisClientes(Clientes *clientes) {
    system("clear || cls");

    Listagens *listagemTopPaisesMaisClientes;

    int contador = 0;

    // criar um array do tamanho dos clientes, para nao ter que fazer realloc depois
    listagemTopPaisesMaisClientes = (Listagens*) malloc(sizeof (Listagens) * clientes->contador);

    // Guardar a informação sem repetir
    for (int i = 0; i < clientes->contador; i++) {
        int existe = 0; // alerta se a variavel existe no array listagemTopPaisesMaisClientes

        for (int j = 0; j < contador; j++) {
            if (strcmp(clientes->cliente[i].origem, listagemTopPaisesMaisClientes[j].nome) == 0) { // caso exista
                listagemTopPaisesMaisClientes[j].contador++;
                existe = 1;
                break;
            }
        }

        if (!existe) { // caso nao exista
            strcpy(listagemTopPaisesMaisClientes[contador].nome, clientes->cliente[i].origem); // copia o nome do pais para a variavel nome do array
            listagemTopPaisesMaisClientes[contador].contador = 1; // como encontrou um pais novo, entao coloca o contador a 1
            contador++; // vai incrementar o contador que serve de inidce do array listagemTopPaisesMaisClientes
        }
    } //Cria e preenche os paises e a quantidade que os mesmos se repetem

    ordenarListagens(listagemTopPaisesMaisClientes, contador);

    printf("Paises com mais clientes registados: \n");
    printArrayListagens(listagemTopPaisesMaisClientes, contador);

    free(listagemTopPaisesMaisClientes);
    listagemTopPaisesMaisClientes = NULL;

}
```

Figura 6 - Função Listagem dos Países com mais Clientes

- ## Países de origem dos clientes que têm mais encomendas

A listagem anterior era importante, pois é de extrema relevância para uma empresa, saber para onde vendem mais os seus produtos, mas, não é porque existem mais clientes de um determinado país, que esse país de origem em questão seja aquele que mais rende para a empresa, pois um cliente pode requerer quantas encomendas quiser , logo, um cliente, ou um pequeno conjunto de clientes, pode requerer mais encomendas, do que todos os clientes do país de origem mais frequente. Portanto, este programa conta com uma função que lista e ordena por ordem decrescente os países de origem dos clientes que têm mais encomendas.

Esta função começa por alocar memoria para um novo array dinâmico de estruturas chamado "listarPaisesMaisEncomendas" do tamanho do número de clientes existentes na estrutura "Clientes". Depois itera[4] através dos clientes e adiciona informações sobre o país de origem e a quantidade de encomendas do país no array "listarPaisesMaisEncomendas", sem repetir países já existentes no array. A função "ordenarListagens" é então chamada para ordenar o array "listarPaisesMaisEncomendas" e a função "printArrayListagens" é chamada para imprimir o array. No final, liberta a memoria alocada para o array criado previamente.

```
void listarPaisesMaisEncomendas(Clientes *clientes) {
    system("clear || cls");

    Listagens *listarPaisesMaisEncomendas;

    int contador = 0;

    // é criado um array do tamanho dos clietens, para nao ter que fazer realloc depois
    listarPaisesMaisEncomendas = (Listagens*) malloc(sizeof(Listagens) * clientes->contador);

    // Guardar infomormação sem repetir
    for (int i = 0; i < clientes->contador; i++) {
        int existe = 0; // alerta se a variavel existe no array recorrente

        for (int j = 0; j < contador; j++) { // caso exista
            if (strcmp(clientes->cliente[i].origem, listarPaisesMaisEncomendas[j].nome) == 0) {
                listarPaisesMaisEncomendas[j].contador += clientes->cliente[i].quantidadeEncomendas;
                existe = 1;
                break;
            }
        }
        if (!existe) { // caso nao exista
            strcpy(listarPaisesMaisEncomendas[contador].nome, clientes->cliente[i].origem); // copia o nome do pais para a variavel nome do array
            listarPaisesMaisEncomendas[contador].contador = clientes->cliente[i].quantidadeEncomendas;
            contador++;
        }
    } //Cria e preenche os paises e a quantidade de encomendas de encomendas soamadas do respetivos clientes

    ordenarListagens(listarPaisesMaisEncomendas, contador);

    printf("Lista dos Paises com mais encomendas: \n");
    printArrayListagens(listarPaisesMaisEncomendas, contador);

    free(listarPaisesMaisEncomendas);
    listarPaisesMaisEncomendas = NULL;
}
```

Figura 7 - Listagem dos países com o maior número de encomendas

[4] repete

- ## Clientes com mais encomendas

Uma informação que é muito relevante para uma empresa, é saber quem são os seus melhores clientes, logo, foi implementado no programa uma função que apresenta os clientes de ordem decrescente, onde em cima estão os clientes que realizaram mais encomendas, e em baixo, os clientes que realizaram menos encomendas.

Esta função recebe como parâmetro a variável produtos, e dentro dela, vai ordenar todos os produtos pela quantidade de vezes que este é vendido numa encomenda, e vai guardá-los por ordem decrescente. Ela vai guardar um produto na variável swap, e vai percorrer todo o array, até encontrar um produto que foi mais vendido, caso haja, vai trocar com o produto que tinha guardado. Quando acabar de percorrer o array, vai guardar o próximo produto na variável swap e voltar a percorrer todo o array. Assim sucessivamente até verificar todos os produtos.

Após a função ordenar os clientes, é só utilizar a função "listarClientes" para que os clientes sejam apresentados na consola com a ordem pretendida.

```
void ordenarClientesPorQntEncomenda(Clientes * clientes) {
    int i, j, position;

    Cliente swap;

    for (i = 0; i < (clientes->contador - 1); i++) {
        position = i;
        for (j = i + 1; j < clientes->contador; j++) {
            if (clientes->cliente[position].quantidadeEncomendas < clientes->cliente[j].quantidadeEncomendas) {
                position = j;
            }
        }
        if (position != i) {
            swap = clientes->cliente[i];
            clientes->cliente[i] = clientes->cliente[position];
            clientes->cliente[position] = swap;
        }
    }
}

void clientesComMaisEncomendas(Clientes * clientes) {
    system("clear || cls");

    ordenarClientesPorQntEncomenda(clientes);

    listarClientes(clientes);
}
```

Figura 8 - Funções que ordenam e listam os clientes por quantidade de encomendas

- ## Materiais mais usados

Para poder confecionar um produto, são precisos vários materiais, portanto, um material pode ser usado para confecionar vários produtos, logo, é de grande relevância para a empresa poder saber quais são os materiais mais usados, assim, saberá quais os materiais para comprar em maior quantidade.

Esta função funciona em conjunto com outra função chamada "ordenarMateriaisPorUsados", que vai ordenar os materiais de acordo com o número de vezes que eles foram usados. Ela faz isso ao comparar cada elemento com todos os outros elementos e trocando-os de posição se necessário.

```
void ordenarMateriaisPorUsados(Materiais * materiais) {
    int i, j, position;
    Material swap;

    for (i = 0; i < (materiais->contador - 1); i++) {
        position = i;
        for (j = i + 1; j < materiais->contador; j++) {
            if (materiais->material[position].vezesUsadas < materiais->material[j].vezesUsadas) {
                position = j;
            }
        }
        if (position != i) {
            swap = materiais->material[i];
            materiais->material[i] = materiais->material[position];
            materiais->material[position] = swap;
        }
    }
}

void materiaisMaisUsados(Materiais * materiais) {
    system("clear || cls");

    ordenarMateriaisPorUsados(materiais);
    listarMateriais(materiais);
}
```

Figura 9 - Funções que ordenam e listam os materiais pelos mais usados

- Produtos mais vendidos

Esta listagem está diretamente interligada com a listagem dos materiais mais usados, pois é importante para a empresa saber quais são os materiais mais utilizados, mas não importa saber onde adquirir mais materiais, sem saber onde eles serão usados, logo é relevante haver uma listagem como esta no programa, para a firma saber onde deverá investir mais..

Esta função funciona em conjunto com outra função chamada "ordenarProdutosPorVendidos", que vai ordenar os produtos de acordo com o número de vezes que eles foram vendidos. Essa função faz isso ao comparar cada item com todos os outros e trocando-os de posição quando necessário.

```
void ordenarProdutosPorVendidos(Produtos *produtos) {
    int i, j, position;
    Produto swap;

    for (i = 0; i < (produtos->contador - 1); i++) {
        position = i;
        for (j = i + 1; j < produtos->contador; j++) {
            if (produtos->produto[position].vezesUsadas < produtos->produto[j].vezesUsadas) {
                position = j;
            }
        }
        if (position != i) {
            swap = produtos->produto[i];
            produtos->produto[i] = produtos->produto[position];
            produtos->produto[position] = swap;
        }
    }
}

void produtosMaisVendidos(Produtos * produtos, Materiais * materiais) {
    system("clear || cls");

    ordenarProdutosPorVendidos(produtos);
    listarProdutos(produtos, materiais);
}
```

Figura 10 - Funções que ordenam e listam os produtos pelos mais vendido

Criação de códigos aleatórios

Uma das primeiras coisas a serem criadas no projeto, foi uma função que cria um código aleatório de sete caracteres, sendo que o caractere inicial será sempre aquele que foi especificado no índice da função, esse caractere inicial permite ao utilizador criar códigos distintos, para que seja percetível onde eles pertencem, como por exemplo, se um código começar com a letra C, significa que é um código de cliente, caso comece com a letra E, então é um código de encomenda. Este caractere inicial, para além de permitir o utilizador identificar o código em questão, também permite um leque maior de possibilidades de código, fazendo com que seja muito improvável que seja criado um código repetido, e mesmo que isso aconteça, as funções vão ver que o código que a função retornou já existe, e vai pedir para que seja criado um novo código, até ser estabelecido um código que não exista.

```
void criarCodigo(char tipoCod, char *array, int numCaracteres) {
    // TipoCod e a variavel que e usada no comeco do codigo e ela serve para identificar o tipo do codigo

    char alfabeto[] = "abcdefghijklmnopqrstuvwxyz";
    char numeros[] = "0123456789";
    char codigo[numCaracteres];

    codigo[0] = tipoCod; ///< Colocar o caracter inicial

    for (int i = 1; i < numCaracteres - 1; i++) {
        // Gera num entre 0 e 1, 1 = numero; 0 = letra
        int numOuLetra = rand() % 2;

        // Gera letra
        if (numOuLetra == 0) {
            int randLetra = rand() % 26; // Gera letra entre A e F
            codigo[i] = alfabeto[randLetra];
        } else { // Gera numero
            int randNum = rand() % 10; // Gera um numero entre 0 e 9
            codigo[i] = numeros[randNum];
        }
    }
    codigo[numCaracteres] = '\0';

    strcpy(array, codigo);
}
```

Figura 11 - Função que gera código aleatório

```
CodMaterial: M0013
Nome: Fundo 155x35
Quantidade: 3
```

Figura 12 - Exemplo de um código aleatório em um material

Gestão de administradores

Para além das gestões que foram pedidas no enunciado, foi implementada outra que permite um administrador adicionar, remover, alterar listar e exportar outros administradores. Para isso, foram adicionadas novas estruturas, sendo elas:

```
typedef struct {
    char codAdministrador[INDICE_COD_ADMINISTRADOR];
    char nome[INDICE_NOME_ADMINISTRADOR];
    char password[INDICE_PASSWORD_ADMINISTRADOR];
}Administrador;

typedef struct {
    int contador;
    Administrador *administrador;
    int tamanhoArray;
}Administradores;
```

Figura 13 - Estruturas dos Administradores

Esta estrutura é muito parecida com as estruturas dos materiais e dos clientes. Mas nestas, foram usadas as seguintes variáveis: o código do administrador; o nome do mesmo e a sua password. O administrador para poder entrar com a sua conta, precisa colocar o seu código e a sua password.

Pesquisas pelo nome

Tirando a gestão de encomendas e a gestão de produções, uma das opções que aparece nos menus, é a pesquisa pelo nome. Essa opção foi feita pois, como a inserção de valores que já existem é sempre feita a partir do código desse elemento, as funções de listar acabam por ser cruciais para o manuseamento do programa, mas caso o programa tenha muita informação já armazenada, tentar encontrar o elemento que se quer no meio da listagem pode-se tornar uma tarefa frustrante e desnecessária, por isso foram feitas pesquisas pelo nome, onde basta apenas inserir o nome, que é algo muito mais fácil de decorar, e depois será apenas exibida a informação referente a esse mesmo elemento. Caso hajam vários elementos com o mesmo nome, acabarão por ser apresentados múltiplos elementos.

```
void procurarNomeProduto(const Produtos *produtos, const Materiais *materiais, char *nome) {
    system("clear || cls");

    int verificacao = 0;

    for (int i = 0; i < produtos->contador; i++) {
        if (strcmp(produtos->produto[i].nome, nome) == 0) {
            imprimirProduto(&produtos->produto[i], materiais);
            verificacao++;
        }
    }

    if (verificacao == 0) {
        printf(ERRO_PRODUTO_INEXISTENTE);
    }
    pause();
}
```

Figura 14 - Exemplo de procura por nome nos produtos

Exportações para CSV

Como foi explicado em cima, como a inserção de valores que já existem é sempre feita a partir do código dos elementos, as opções de listagem e pesquisa são fundamentais para o manuseamento do programa, no entanto, mesmo com essas opções que auxiliam o utilizador, ainda assim acaba por não ser muito prático encontrar elementos dentro do programa, sobretudo quando não se sabe o nome do mesmo, portanto toda a gestão tem uma opção que permite exportar o seu conteúdo para um ficheiro csv, assim podendo abri-lo num ficheiro excel, que é muito mais prático para visualizar o conteúdo e enviar para outra pessoa ver.

```
CodProduto;Nome;Dimensoes;preco;CodMaterial;Descricao;Quantidade;Unidade
P00001;Movel WC ;80x120x60;99.00;M0001;Tubo Cola 10GR;1;UN
;;;;M0002;Parafuso 30MM;4;UN
;;;;M0003;Bucha 30MM;4;UN
;;;;M0004;Base 80x60;2;UN
;;;;M0005;Pés;4;UN
;;;;M0006;Porta 80x120;1;UN
;;;;M0007;Fundo 80x120;1;UN
;;;;M0008;Laterais 60x120;2;UN
;;;;M0009;Cavilhas;24;UN
;;;;M0023;Prateleiras 75x55;3;UN
P00002;Cómoda 3 Gavetas ;160x120x60;299.00;M0001;Tubo Cola 10GR;4;UN
;;;;M0002;Parafuso 30MM;8;UN
;;;;M0003;Bucha 30MM;8;UN
;;;;M0008;Laterais 60x120;2;UN
;;;;M0010;Fundo 160x120;1;UN
;;;;M0011;Base 160x60;2;UN
;;;;M0012;Ferragem 40CM;3;PAR
;;;;M0013;Fundo 155x35;3;UN
;;;;M0014;Laterais 55x35;6;UN
;;;;M0015;Frente 160x38;3;UN
;;;;M0016;Base 155x55;3;UN
P00003;Mesa de Cabeçeira ;60x60x60;149.00;M0001;Tubo Cola 10GR;1;UN
;;;;M0002;Parafuso 30MM;4;UN
;;;;M0003;Bucha 30MM;4;UN
;;;;M0004;Base 80x60;2;UN
;;;;M0005;Pés;4;UN
;;;;M0006;Porta 80x120;1;UN
;;;;M0007;Fundo 80x120;1;UN
;;;;M0008;Laterais 60x120;2;UN
```

Figura 15 - Exemplo de ficheiro csv exportado dos produtos

4. Estrutura analítica do projeto

Planeamento temporal do projeto

Tabela 1 - Planeamento temporal do projeto

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6
Planeamento	X					
Criação das estruturas	X	X				
CRUD's		X	X	X		
Gravar / ler ficheiros			X	X		
Memória Dinâmica			X	X	X	
Criação de menus				X		
Criação de Exportações				X		
Criação de Pesquisas					X	
Gestão de Produções					X	
Criação de Listagens				X	X	X
Importação de produtos						X
Comentários em Doxygen					X	X
Correção de erros	X	X	X	X	X	X
Testes		X	X	X	X	X
Relatório					X	X

Divisão das tarefas entre o grupo

Tabela 2 - Divisão das tarefas entre o grupo

	César	Hugo	Pedro
Estruturas	X	X	X
CRUD's	X	X	X
Pesquisas			X
Exportações		X	
Listagens	X		X
Importação		X	X
Menus	X	X	
Correção de erros	X	X	X
Testes	X		X
Gestão de Produções			X
Doxygen		X	
Relatório		X	X

5. Funcionalidades implementadas

Como o programa tem muitas funcionalidades, e como todas funcionam de uma maneira semelhante, portanto, quando forem apresentadas as funcionalidades que foram implementadas, elas serão descritas de uma forma genérica para apresentar todas as funcionalidades de modo a não repetir informação, e quando forem apresentadas imagens, serão usadas as funções dos clientes, de modo a ilustrar como elas funcionam.

Leitura de dados guardados nos ficheiros

Quando o programa é iniciado, antes de ser apresentado o menu inicial, são importados os dados dos ficheiros guardados para dentro das variáveis, permitindo que os dados não sejam perdidos quando o programa for encerrado.

As funções que leem os dados dos ficheiros, começam por abrir o ficheiro que foi passado na função, caso ele exista, vai ser lida a variável contador desse elemento, para que a seguir seja alocada a memoria necessária para armazenar os dados que serão lidos a seguir, essa alocação é feita com uma margem de vinte espaços na memória, para que não seja preciso ter que estar constantemente a realocar memória sempre que se inserir um novo registo, basta faze-lo de vinte em vinte.

Após ser alocada a memória, é lida e guardada a informação através do comando fread[5].

[5] Lê dados de um fluxo, retorna o número de itens completos lidos pela função

Criar elementos

A criação de elementos acontece a partir de uma função que começa por verificar se o tamanho do array que guarda os dados desse mesmo elemento é igual ao contador do mesmo, caso seja igual, o array é aumentado em vinte.

Após verificar o array onde são guardados os elementos, é criado um código, caso ele já exista, vai ser pedido um novo chamando a mesma função, caso não exista, então será pedidos ao utilizador para que insira os dados que serão salvos no array, no final, o contador de elementos é incrementado em um.

No caso dos produtos e das encomendas, durante a inserção dos dados que serão salvos, é criado um ciclo do while[6], que vai pedir ao utilizador para inserir um código de material, no caso dos produtos, ou um código de produto, no caso das encomendas, caso o código não exista, será pedido um código novo, caso exista, vai pedir a quantidade do respetivo elemento que se adicionou, e vai voltar a pedir para ser inserido um código até que o utilizador escreva "fim" na consola. Isto acontece, pois, um produto pode precisar de múltiplos materiais para ser feito, e uma encomenda pode conter múltiplos produtos em quantidades diferentes.

Quando os materiais estão a ser inseridos, quando é verificado que o código que o utilizador embutiu existe, a memória do respetivo array é realocada em mais um, assim pode-se inserir todos os materiais necessários, não fincando preso a um número pré-definido. Após isso, já dentro do if (estrutura de decisão), vai verificar se é preciso aumentar o array com a função "aumentarArray", caso seja preciso, então o array será realocado em mais 20 espaços.

```
void aumentarArrayClientes(Clientes *clientes) {
    if (clientes->contador >= clientes->tamanhoArray) {
        clientes->tamanhoArray = clientes->contador + MALLOC_MARGEM;
        clientes->cliente = (Cliente *) realloc(clientes->cliente, clientes->tamanhoArray * sizeof (Cliente));
    }
}
```

Figura 16 - Função aumentar array clientes

Após chamar a função "aumentarArray", será pedido ao utilizador para inserir os dados do novo elemento.

[6] Executa a repetição de um bloco de instruções enquanto uma condição é verdadeira.

```
// Aumentar o array clientes, caso o contador dos clientes seja igual ao array clientes
aumentarArrayClientes(clientes);

// Codigo Cliente
strcpy(clientes->cliente[clientes->contador].codCliente, codCliente);

// Nome Cliente
obterString(clientes->cliente[clientes->contador].nome, INDICE_NOME_CLIENTE, MSG_INSERIR_NOME_CLIENTE);

// Morada
obterString(clientes->cliente[clientes->contador].morada, INDICE_MORADA_CLIENTE, MSG_INSERIR_MORADA);

// NIF
clientes->cliente[clientes->contador].nif = obterInt(INDICE_MIN_NIF, INDICE_MAX_NIF, MSG_INSERIR_NIF);

// Origem
obterString(clientes->cliente[clientes->contador].origem, INDICE_ORIGEM_CLIENTE, MSG_INSERIR_PAIS_ORIGEM);

// Password
obterString(clientes->cliente[clientes->contador].password, INDICE_PASSWORD_CLIENTE, MSG_INSERIR_PASSWORD);

// Estado
clientes->cliente[clientes->contador].estado = DESATIVO; /// O estado vem desativo por padrao, e so fica aitvo quando o ciente fazer uma encomenda

// Quantidade de encomendas que o cliente realizou
clientes->cliente[clientes->contador].quantidadeEncomendas = 0;

clientes->contador++;
```

Figura 17 - Inserção de valores nos clientes

Alterar elementos

As funções que alteram os elementos são muito parecidas com as funções adicionar, pois elas pedem para serem inseridos novos valores, a diferença é que, no adicionar, é gerado um código para ser usado, no alterar, é pedido para inserir o código do elemento que se pretende alterar, caso ele não exista, não será possível alterar nada.

Os únicos campos que não é possível alterar são: o código do elemento; o código de cliente, no caso das encomendas; e as passwords, no caso dos clientes, caso esteja a ser alterado por um administrador, pois, somente o próprio cliente é que pode alterar esse campo.

```
int indiceCliente;

indiceCliente = procurarCliente(clientes, codCliente);

if (indiceCliente != -1) {

    // Nome Cliente
    obterString(clientes->cliente[indiceCliente].nome, INDICE_NOME_CLIENTE, MSG_INSERIR_NOME_CLIENTE);

    // Morada
    obterString(clientes->cliente[indiceCliente].morada, INDICE_MORADA_CLIENTE, MSG_INSERIR_MORADA);

    // NIF
    clientes->cliente[indiceCliente].nif = obterInt(INDICE_MIN_NIF, INDICE_MAX_NIF, MSG_INSERIR_NIF);

    /// O administrador não pode alterar a password do cliente, so o proprio cliente pode fazer isso

    // Origem
    obterString(clientes->cliente[indiceCliente].origem, INDICE_ORIGEM_CLIENTE, MSG_INSERIR_PAIS_ORIGEM);

    return 1; // alterou com sucesso

}

return -1; // nao conseguiu alterar pois o cliente nao existe
```

Figura 18 - alterar valores de um cliente

Nas funções de alterar, se o valor retornado for -1, então significa que o código inserido pelo utilizador não existe. Caso o código retornado seja 1, então a alteração foi bem sucedida.

Remover Elementos

As funções que removem um elemento, começam por pedir ao utilizador o código daquilo que o utilizador pretende remover, caso o código não exista, vai saltar fora, caso exista vai fazer um ciclo que passe todos os elementos que estão à frente daquele que se pretende remover um espaço para trás, depois é chamada uma função que liberta a memória do elemento que se quer remover, e coloca os bytes a zero. No final, é decrementado o contador desse mesmo elemento em um.

```
int indiceCliente, i;

indiceCliente = procurarCliente(clientes, codCliente);

if (indiceCliente != -1) {
    if (clientes->cliente[indiceCliente].quantidadeEncomendas == 0) { // Só deve dar para remover clientes que nao tenham realizado encomendas
        for (i = indiceCliente; i < clientes->contador - 1; i++) {
            clientes->cliente[i] = clientes->cliente[i + 1];
        }

        apagarDadosCliente(&clientes->cliente[i]);

        clientes->contador--;

        return 1; // removeu com sucesso

    }

    return 0; // nao removeu pois o cliente ja tinha encomendas
}

return -1; // codigo de cliente nao existe
```

Figura 19 - Função apagar cliente

Nas funções remover, se o valor retornado for -1, então significa que o código inserido pelo utilizador não existe. Caso o código retornado seja 1, então a remoção foi bem sucedida, e no caso dos clientes, se o valor retornado for 0, então não foi possível remover o cliente, pois este já fez alguma encomenda.

Listar Elementos

Estas funções permitem aos utilizadores consultarem os dados que foram inseridos no programa. Estas listagens podem variar de estrutura para estrutura, pois, no caso dos clientes e das encomendas, é possível listar apenas os clientes ou encomendas que estejam com o estado ativo ou inativo, mas tirando essas duas, todas as estruturas têm uma listagem que apresenta ao utilizador todos os elementos dessa estrutura sem nenhum tipo de filtro.

```
Cliente 1:
        CodCliente: C3doyf1
        Nome: Alfredo
        Morada: rua do sobreiro
        Nif: 123456789
        Pais de Origem: portugal
        Estado: Ativo

Cliente 2:
        CodCliente: C5du27e
        Nome: Roberto
        Morada: rua das moreiras
        Nif: 123456888
        Pais de Origem: espanha
        Estado: Ativo

Cliente 3:
        CodCliente: Cb2t7a0
        Nome: espanha
        Morada: espanha
        Nif: 123456789
        Pais de Origem: espanha
        Estado: Desativo
```

Figura 20 - output da listagem de clientes

Procurar Elementos

Tirando a gestão de encomendas e a gestão de produções, uma das opções que aparece nos menus, é a pesquisa pelo nome. Essa opção foi feita pois, como a inserção de valores que já existem é sempre feita a partir do código desse elemento, as funções de listar acabam por ser cruciais para o manuseamento do programa, mas caso o programa tenha muita informação já armazenada, tentar encontrar o elemento que se quer no meio da listagem pode-se tornar uma tarefa frustrante e desnecessária, por isso, foram criadas pesquisas pelo nome, onde basta apenas inserir o nome, que é algo muito mais fácil de decorar, e depois será apenas exibida a informação referente a esse mesmo elemento. Caso hajam vários elementos com o mesmo nome, acabarão por ser apresentados múltiplos elementos.

```
int verificacao = 0;

for (int i = 0; i < clientes->contador; i++) {
    if (strcmp(clientes->cliente[i].nome, nome) == 0) {
        imprimirCliente(&clientes->cliente[i]);
        verificacao++;
    }
}

if (verificacao == 0) {
    printf(ERRO_CLIENTE_INEXISTENTE);
}
```

Figura 21 - Função de procurar um cliente pelo nome

Na figura 22, mostra o que é exibido na consola caso se insira o nome Alfredo quando é pedido. Como a pesquisa é feita pelo nome, pode haver vezes que mostre mais que um elemento, neste caso um cliente, pois podem haver vários clientes com o mesmo nome.

```
CodCliente: C3doyf1
Nome: Alfredo
Morada: rua do sobreiro
Nif: 123456789
Pais de Origem: portugal
Estado: Ativo


Clique no enter para continuar.
```

Figura 22 - output da função de procurar cliente pelo nome

Exportar Elementos

A exportação de elementos é feita para um ficheiro csv, que é basicamente um ficheiro onde a informação é separada por virgulas, logo, nas funções que exportam os dados para o ficheiro csv, elas primeiramente criam o ficheiro e inserem os títulos na primeira linha, depois fazem um ciclo que inserem os dados no ficheiro seguindo a ordem dos títulos colocados no começo.

```
CodCliente;Nome;Morada;Nif;Origem
C3doyf1;Alfredo;rua do sobreiro;123456789;portugal
C5du27e;Roberto;rua das moreiras;123456888;espanha
```

Figura 23 - ficheiro clientes.csv exportado do programa

Guardar Elementos

As funções que guardam os elementos são bastante semelhantes com as funções que os leem, só que nestas, usa-se a função fwrite[7] para escrever um ficheiro binário com toda a informação do programa. As funções começam por guardar o contador e depois os dados das estruturas, caso haja um array dentro da estrutura, como nas encomendas e nos produtos, é criado um ciclo que escreve no ficheiro toda a informação desse array.

```
FILE *fp = fopen(ficheiro, "wb");

if (fp == NULL) {
    perror("\n\nNao foi possivel guardar os clientes!");
    pause();
    return;
}

fwrite(&clientes->contador, sizeof (int), 1, fp);

for (int i = 0; i < clientes->contador; i++) {
    fwrite(&clientes->cliente[i], sizeof (Cliente), 1, fp);
}

fclose(fp);
```

Figura 24 - Função guardar clientes

[7] Escreve um bloco de dados num ficheiro, e retorna o número de itens escritos

Importação de Produtos

Uma das funções que foi pedida no enunciado do projeto, foi converter um ficheiro excel para o formato que queríamos, e criar uma função que lia esse ficheiro e colocasse os seus dados nos arrays do programa.

O formato que foi escolhido para a conversão do excel foi o csv, logo, criou-se uma função que lê o ficheiro csv especificado pelo caminho introduzido pelo próprio utilizador, e guarda os dados no array dos produtos, e no array dos materiais. Sendo que, não irá inserir os produtos ou os materiais que já existem no programa, vai apenas carregar apenas aqueles que não existem ainda.

6. Conclusão

Ao olhar em retrospetiva, achamos que este trabalho foi concluído com sucesso, que todos os parâmetros e funcionalidades propostas foram concluídas com êxito, para além de conseguirmos colocar algumas funcionalidades que não foram pedidas, fazendo assim, o projeto ficar muito mais completo.

Em suma, este projeto foi muito importante para o nosso desenvolvimento, pois conseguimos colocar em prática aquilo que foi lecionado durante as aulas de fundamentos e laboratório de programação, fazendo com que sedimentássemos os nossos conhecimentos em C, que com certeza serão úteis no futuro.

7. Bibliografia

http://linguagemc.com.br/enum-em-c/

http://www.bosontreinamentos.com.br/programacao-em-linguagem-c/arrays-em-c-declaracao-inicializacao-e-atribuicao-de-valores/

https://pt.stackoverflow.com/questions/179205/qual-%C3%A9-a-diferen%C3%A7a-entre-calloc-e-malloc

https://learn.microsoft.com/pt-br/cpp/c-runtime-library/reference/fread?view=msvc-170

http://linguagemc.com.br/o-comando-while-em-c/