ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Projeto de Laboratório de Programação

Licenciatura em Engenharia Informática

Licenciatura em Segurança Informática em Redes de Computadores

2022/2023

Grupo 200

David Santos – 8220651

Diogo Nogueira – 8220726

Leandro Afonso – 8220302

Índice

1. Introdução

No âmbito da Unidade Curricular (UC) Laboratório de Programação, foi-nos solicitado a realização de um projeto final que objetiva avaliar os alunos com base nos conhecimentos adquiridos nas UC de Laboratórios de Programação (LP) e Fundamentos de Programação (FP).

Este projeto tem como objetivo geral o desenvolvimento de uma aplicação de registo de encomendas cumprindo os requisitos e funcionalidades exigidas num prazo máximo de 7 semanas a partir do dia em que foi anunciado (05/12/2022). Já os objetivos específicos são os seguintes:

- Especificar e coordenar um projeto em grupo de pequena dimensão;
- Compreender e dominar os conhecimentos teóricos e práticos sobre algoritmia e programação na linguagem C;
- Adquirir competências com vista à resolução de problemas, nomeadamente através da pesquisa e utilização autónoma de conteúdos e ferramentas externas;
- Utilizar o desenvolvimento de um projeto de desenvolvimento de software de pequena/média dimensão como elemento essencial do processo de aprendizagem individual.

O projeto baseia-se no desenvolvimento de uma aplicação de registo de encomendas para a empresa "**Móveis para Todos**". Trata-se de uma aplicação capaz de registar, alterar e eliminar dados referentes a cada cliente, registar, alterar e eliminar encomendas, bem como exportar para ficheiro um resumo dos componentes a usar para satisfazer as encomendas de uma determinada semana.

De forma a alcançar estes objetivos, o grupo 200 começou por estudar os *apontadores* (e como funcionam), para depois implementar em *structs e memória dinâmica,* que também foram outros tópicos estudados. Desde o início do projeto foi tentado manter uma estrutura "limpa" de pastas e ficheiros do projeto de forma a alcançar uma melhor navegação e prevenir consumo desnecessário de tempo. Este assunto de arquitetura de pastas limpa e bem-estruturada foi bem explicito do início ao fim do projeto, o que fez com que poupássemos muito tempo para produtividade.

2. Funcionalidades requeridas

Gestão de clientes

A gestão de clientes é uma contem um conjunto de funcionalidades cruciais para este programa. Ele permite registar, alterar, procurar e eliminar dados referentes a cada cliente, nomeadamente: código de cliente, nome, morada, NIF, país, entre outros.

Estrutura de clientes

A *struct Customer* contém o *int id*, a *string name* que guarda o nome do cliente, *string role,* o *int NIF*, a *string country* para país, e a *struct Address* para morada que inclui as *strings city* e *street*, e o *int nDoor* (número de porta).

Já a *struct Customers* consiste numa lista de clientes. Sendo assim, ela possui um apontador para *struct Customer* e dois *int* correspondentes ao número de clientes presentes na lista (*nCustomers*) e ao número máximo de clientes para a lista(*nMax*).

Listar clientes - listCustomers

A função *listCustomers* que recebe como parâmetro um *struct Customers*, começa por verificar se o número total de clientes existentes (*Customers.nCustomers*) é maior que 0. Se a condição for verdadeira, acontece um ciclo "*for*" que executa a função *printCustomer* por *n* vezes, sendo *n*, o número de clientes (*Customers.nCustomers*). Se a condição for falsa, é mostrado uma mensagem de erro no terminal informando da inixistência de clientes na lista.

Listar um cliente - printCustomer

A função *printCustomer* dá output com uma função *printf* a todos os dados da *struct Customer* que é recebida como parâmetro da função.

Verificar se o cliente existe - checkCustomer

A função *checkCustomer* recebe como parâmetros o *struct Customers* e o *int id* do cliente a ser verificado. A função tem um ciclo "*while*" que corre enquanto a variável local *"i = 0"* é menor que o número de clientes (*Customers.nCustomers*) e, dentro do ciclo, há uma condição que retorna o valor de *"i"* (referente à posição do ID) se o valor de *Customers.customers[i].id* equivaler o valor do *id* recebido como parâmetro. Se nenhum cliente for encontrado. A função retorna *-1*.

Pesquisar cliente - searchCustomer

A função *searchCustomer* recebe como parâmetros o *struct Customers*. A função declara uma variável *pos* (posição) que é o valor retornado da função checkCustomer, onde o *id* de checkCustomer é um prompt pedido ao utilizador. Se o cliente for encontrado é mostrado no terminal.

Alterar (ou não) dados do cliente - editCustomer

A função tem como parâmetros o *struct Customers*, uma *string filename* e o *int id* do cliente. Inicialmente, é verificado se o cliente existe a partir do id. Se o cliente existir, são temporariamente reescritos os valores de cada campo do cliente e, de seguida, é pedida ao usuário a confirmação para guardar os novos valores na base de dados.

Editar cliente - updateCustomer

Tendo como parâmetros o *struct Customers*, uma *string filename* e um *int pos* (posição do cliente a editar), a função coloca o cursor no inicio do ficheiro e depois na linha relacionada à posição do cliente, e reescreve a linha toda com os novos dados do cliente.

Eliminar cliente - deleteCustomer

A função recebe como parâmetros o *struct Customers*, *struct Orders*, *string filename*, e um *int id.* Começa por verificar se o id é de um administrador. Sendo de administrador, o utilizador pode escrever um id de cliente para o mesmo ser eliminado. Se não existir nenhum cliente com o id, é mostrado um erro no terminal. Não sendo de administrador, o id usado é do próprio utilizador. Depois, é verificado se o cliente a ser eliminado tem encomendas ativas. Se tiver, só é possivel a mudança de Estado do cliente para Inativo, sendo perguntado uma confirmação de alteração do estado. Se o cliente não tiver encomendas, é perguntada a confirmação de eliminação do cliente.

Se o cliente for eliminado, é feito um *realocamento de memória*.

Gestão de produtos

A gestão de produtos permite editar e remover produtos a comercializar bem como procurar e listar dados referentes a cada produto, nomeadamente o nome, dimensões e preço.

Estrutura de produtos

A *struct Product* contém o *int id*, a *string name*, a *string dimensions* e o *int* preço, tal como o *boolean isActive*.

A *struct Products* consiste numa lista de produtos. Esta possui um *apontador* para a *struct Products* e dois *int*, sendo estes o número de produtos presentes na lista (*nProds*) e o número máximo de produtos para a lista (*nMax*).

Listar produtos – listProducts

A função *listProducts* recebe como parâmetro o struct Products e começa por validar se o número total de produtos existentes (*Products.nProds*) é maior que 0. Sendo esta condição verdadeira, é iniciado um *"for loop"* que executa a função printProducts *n* vezes, sendo *n* o número de produtos (*Products.nProds*). Já se a condição for falsa, é apresentado um erro no terminal.

Listar um produto- printProducts

A função *printProducts* apresenta todos os membros do *struct Product*, recorrendo a uma função *printf,* que é recebido como parâmetro da função.

Verificar se o produto existe – checkProduct

A função *checkProduct* recebe como parâmetros o struct Products e o id do produto a ser verificado. Esta função tem um "*while loop"* que é executado enquanto a variável local "*i"* é menor que o número de produtos (*Products.nProds*), tendo este *loop* uma condição que retorna o valor de "*i" (*ID) caso o valor de *Products.products[i]* for equivalente ao valor do ID recebido como parâmetro. Na instância em que nenhum seja encontrado, a função retorna *–1.*

Pesquisar produto – searchProduct

A função *searchCustomer* recebe como parâmetro o *struct Products*. Esta função declara uma variável *pos* que é o valor retornado pela função *checkProducts*, em que o *id* de *checkProduct* é um prompt pedido ao utilizador. Se o produto for encontrado, este é apresentado no terminal.

Alterar produto – editProduct

A função recebe como parâmetros o *struct Product*s, uma *string filename* (products.bin) e o *ID* do produto. Esta função começa por verificar se o produto existe através do ID, usando a função *checkProduct*. Se o produto existir, é pedido ao utilizador para inserir um novo nome, novas dimensões e um novo preço, sendo também pedido o estado do produto. Caso o produto não exista, é apresentada uma mensagem de erro no terminal.

Remover produto a comercializar - deleteProduct

A função *deleteProduct* recebe como parâmetros os *structs Products, Orders e Materials*. Esta função pede ao utilizador um ID, inicializa uma variável *pos,* que será o valor retornado pela função *checkProduct,* depois de verificar se existe um produto com esse mesmo ID. Caso o produto exista, é pedido ao utilizador confirmação para a alteração do estado ou eliminação. No caso de existirem encomendas desse produto, este é apenas inativo, se não existirem encomendas, o produto é removido.

Se o produto for eliminado, é chamada a função *reallocProductsAfterDelete,* que decrementa o número de produtos existentes (*Products.nProds)* e realoca a memória.

A função *deleteProductData* também é chamada para desalocar qualquer a memória usada pelo produto.

Esta função atualiza também os *arrays* de materiais e de encomendas de maneira a remover o prooduto.

Finalmente, a função *removeProduct* é chamada para remover os dados do produto do ficheiro e informa o utilizador que o dito produto foi apagado com sucesso.

Gestão de encomendas

Sendo um programa de registo de encomendas, a gestão de encomendas trata-se de um conjunto de funcionalidades essenciais para cumprir com o objetivo do projeto. Ela permite registar, alterar, procurar, listar e eliminar encomendas. As encomendas são feitas baseando na tabela de produtos pré-definida que foi enviada como conteúdo de suporte para este projeto.

As funcionalidades de gestão de encomendas podem ser acessadas pelo administrador, escolhendo a segunda opção “2 – Gestão de encomendas” presente no menu de administração.

Estrutura de encomendas

A *struct Order* (para cada encomenda) é composta pela *string id*, o *array products* que armazena a quantidade para cada um dos produtos, o *int total,* que é a soma do preço de todos os produtos encomendados, o *int customerId* que é o id do cliente que registou a encomenda e pela *struct Date*. Esta, por sua vez, é composta pelos *int day, month* e *year* correspondendo à data de entrega da encomenda.

Já a *struct Orders* diz respeito à lista de encomendas. Ela é composta por um apontador para a *struct Order* e dois *int*: *nOrders* referente à quantidade de encomendas da lista e o *nMax* referente ao máximo de encomendas.

Verificar encomenda - checkOrder

A função *checkOrder* é crucial visto que é invocada na maioria das funcionalidades de gestão de encomendas. Ela recebe como parâmetros a *struct Orders* e uma *string id* a ser verificado. Esta função é encarregue de verificar a existência de uma encomenda procurando pelo Id fornecido na lista. Para fazer isso, é executado um ciclo que percorre as posições da lista de encomendas verificando se o Id de cada posição equivale ao Id fornecido. Caso forem equivalentes, é retornada a posição em que a verificação teve resultado verdadeiro. Caso o final da lista de encomendas for atingido e não foram encontradas nenhuma igualdade, é retornado o valor -1. Esse valor representa para o nosso programa a inexistência do que estava a ser procurado.

Listar encomendas – listOrders

A função *listOrders*, invocada escolhendo a primeira opção do menu de gestão de encomendas, recebe como parâmetro o *apontador* para a *struct Orders* e o *apontador* para a *struct Products*. Caso a lista de encomendas não estiver vazia, ou seja, *nOrders* (número de encomendas) for maior que 0, executa-se em ciclo, partindo do inteiro i=0, que ocorre até que o último número menor que *nOrders* seja atingido. Para cada número valor do inteiro i, é invocada a função *printOrders* enviando como argumento a *struct Order* correspondente à posição "*i*" e a *struct Products*.

Caso a lista de encomendas estiver vazia, é feito o output da mensagem que informa sobre isso.

printOrders

Trata-se da função responsável pelo output dos dados referentes à *struct* de uma encomenda. Ela recebe como parâmetros a *struct Order*, que diz respeito aos dados de uma encomenda e um *apontador* para a *struct Products*.

Primeiramente é feito o output do *id* da encomenda. De seguida é executado um ciclo que corre todas as posições do *array products* da *struct Order*. Caso a posição contenha um número superior a 0, ou seja, caso produto associado a essa posição estiver incluído nessa encomenda, faz-se o print da quantidade encomendada e o nome do respetivo produto, encontrado em *products->products[i].name* (na mesma posição que o *order.products[i]*). Depois de fazer output da quantidade referente aos produtos encomendados, faz-se o output do preço total da encomenda em euros, da data de entrega e do Id do cliente que registou a encomenda.

Registar Encomenda - createOrders

A função *createOrders* é invocada selecionando a segunda opção do menu de clientes. Trata-se de uma função exclusiva do menu de clientes sendo apenas acessível caso se trate de um cliente logado pois, ao criar uma encomenda, ela é associada com o cliente que fez o seu registo. Esta função recebe como parâmetros: e o nome referente ao ficheiro que armazena a lista de encomendas.

Passando para a codificação, primeiramente é verificado se a quantidade de encomendas registadas (*nOrders*) é igual à quantidade máxima para a lista (*nMax*). Caso essa igualdade for verdadeira, é invocada a função de *realocação* da lista de encomendas aumentando o número máximo de encomendas que podem ser registadas e possibilitando a criação de mais encomendas. De seguida, é invocada a função *createOrder* enviando como argumentos: o

apontador para a *struct Orders*, outro para a *struct Products* e o *int id* do cliente associado à encomenda que será registada.

createOrder

Esta função é que realmente faz o registo de uma encomenda. Ela inicia-se verificando se a quantidade de produtos (*nProds*) presentes na lista de produtos é maior que zero, ou seja, se há produtos para serem encomendados.

Caso existirem produtos na lista, é executado a função *printActiveProds* e o valor retornado é atribuído ao inteiro *activeProds*.

printActiveProds é uma função responsável por fazer output dos produtos da lista de produtos que estão ativos, ou seja, que podem ser encomendados. Sendo assim, ela recebe como parâmetros a lista de produtos. É executado um ciclo que percorre toda a lista de produtos e caso esse produto estiver ativo, faz output dos dados referentes a esse produto e vai contabilizando a quantidade de produtos ativos na variável counter. Depois do ciclo, é verificada se o counter é ou não igual a zero. Se ele for igual a zero é retornado o valor zero e se ele for maior que zero é retornado o seu valor.

Se a condição de activeProds é maior que zero for aprovada, inicia-se o processo de registo de uma encomenda. Começa-se pela criação de um id que consiste na soma de uma unidade com a quantidade de encomendas da lista. Esse *id* passa por um processo de formatação adicionando zeros à frente e a letra 'E' simbolizando encomenda. No final desse processo, o *id* fica por exemplo, com o seguinte formato E00001. Depois, inicializam-se todas as posições do *array orders.products*, referente à quantidade encomendada de cada produto, a zero, é atribuido o valor zero ao total e o *id* do cliente ao *customerId*. Posteriormente, é executado um ciclo que corre enquanto a opção selecionada for diferente de zero que é a opção de finalizar encomenda. Nesse ciclo faz-se o registo da quantidade referente aos produtos ativos que foram apresentados. As possíveis opções vão de zero até o valor *activeProds* referente à quantidade de produtos que foram apresentados. Supondo que temos no máximo 3 opções de produtos, selecionando a primeira opção o programa verifica qual foi o primeiro produto apresentado, pede ao cliente a quantidade que deseja desse produto e armazena na posição referente a esse produto no *orders.products*. O mesmo acontece para as outras opções caso houver mais produtos ativos. Depois de escolher o(s) produto(s) e a quantidade, ao finalizar a encomenda, é invocada a função *calculateTotal* e caso o total for maior que zero, é invocada a função *getDeliveryDate* e retornado a quantidade atualizada de encomendas.

Caso contrário, é retornado o valor -1.

De volta à função *createOrders*, com o inteiro pos igual ao valor retornado em cima, se ele for diferente de -1, é invocada a função *printOrders* para fazer *output* dos dados da encomenda registada e de seguida é perguntado ao cliente se pretende confirmar a encomenda. Caso a resposta seja afirmativa é invocada a função *insertOrders* que faz a adição dessa encomenda no ficheiro da lista de encomendas. Caso contrário, é invocada a função *deleteOrderData* para eliminar os dados referentes a encomenda não confirmada e subtrair uma unidade da quantidade de encomendas (*nOrders*) da lista.

Editar encomenda - editOrder

A função de *editOrder*, invocada selecionando a segunda opção do menu de gestão de encomendas, recebe como parâmetros: o apontador para estrutura de lista de encomendas, outro para a lista de produtos e o nome do ficheiro contendo a lista de encomendas. Esta função

é equivalente à função *createOrder* diferenciando-se em alguns detalhes. Ela pede o *Id* da encomenda que o administrador pretende editar e verifica se esse *Id* corresponde ao *Id* de alguma encomenda na lista de encomendas. Se corresponder, as edições serão realizadas na posição da lista de encomendas que contem o *Id* escolhido. Ao editar os dados da encomenda, é calculado o total novamente com a função *calculateTotal* e é gerado uma nova data de entrega para essa encomenda. Por fim, caso o administrador confirmar as alterações, é invocada a função *updateOrder* que faz a atualização/modificação dos dados referentes a aquela encomenda no ficheiro da lista.

Eliminar encomenda - deleteOrder

A função de deleteOrder, invocada selecionando a terceira opção do menu de gestão de encomendas, recebe como parâmetros: o apontador para o *struct Orders orders* e o nome do ficheiro *orders.bin*, responsável por armazenar os dados referentes à lista de encomendas. Esta função é responsável por eliminar uma determinada encomenda. Primeiramente é pedido ao administrador o Id da encomenda que pretende eliminar. Como vimos anteriormente, o programa que gera o Id da encomenda do seu registo. O administrador só deve inserir o número presente no *ID* e não necessariamente o *id* da encomenda exato, pois o programa faz a sua formatação para o formato de *Id* pré-definido. Com o *Id* já formatado, é criado o inteiro *pos (position)* cujo valor é o número retornado pela função *checkOrder*, enviando como argumento a estrutura da lista de encomendas e o *Id* que se pretende eliminar. Esse inteiro é determinante para esta função visto que ele se trata da posição da encomenda na lista de encomendas (se o *id* da encomenda escolhida acima existir).

Caso essa encomenda existir, ou seja, caso a posição for diferente de -1, é perguntado ao administrador se pretende guardar/confirmar o procedimento. Se a resposta for afirmativa, é executado um ciclo cuja função é apagar os dados referentes à encomenda selecionada utilizando a sobreposição dessa posição pela próxima encomenda. Sendo assim, os dados da encomenda a seguir passam para a posição anterior até que a última posição seja atingida. Durante esse processo, para não causar em saltos ou lacunas de *ID* de encomendas, os *IDs* referentes a cada posição da estrutura de lista de encomendas mantêm-se, sendo sempre o valor da posição + 1. Seguindo este método, as duas últimas posições ficam com os mesmos dados, por isso, depois do ciclo, é invocada a função *deleteOrdersData* enviando como argumento um apontador para a última posição da estrutura de lista de encomendas para que os dados sejam apagados. Em seguida, é feita a redução de uma unidade no *nOrders* (número de encomendas) e é invocada a função *removeOrders* responsável por fazer as alterações no ficheiro *orders.bin*.

Se a encomenda escolhida não existir, ou seja, se o inteiro pos for igual a -1, é feito o output de uma mensagem informando sobre a inexistência dessa encomenda.

Procurar encomenda – searchOrder

A função searchOrder, invocada quando é escolhida a opção "4 – Procurar encomenda" do menu de gestão de encomendas, recebe como parâmetros a estrutura de lista de encomendas Orders orders e o apontador para a estrutura de lista de produtos Products products.

Ela é responsável pela procura e apresentação de uma encomenda escolhida pelo administrador, caso esta exista.

Esta função pede, primeiramente, o *ID* da encomenda e faz a verificação da sua existência e posição na lista de encomendas invocando a função checkOrder e atribuindo o valor retornado para o inteiro *pos* (posição). Caso a encomenda existir e a sua posição for encontrada, é invocada

a função *printOrder*, previamente mencionada, enviando como argumento a posição da encomenda escolhida na lista de encomendas. Caso contrário, é feito o output da mensagem de notificação de inexistência de tal encomenda na lista de encomendas.

Gestão de materiais

A gestão de materiais permite listar, editar e eliminar os materiais pré-definidos no conteúdo de suporte. Para aceder ao menu de gestão de materiais, seleciona-se a quarta opção do menu de administração.

Estrutura de materiais

A estrutura de material (*Material*) contém o ID (*int*), a descrição (*string*), a quantidade desse material referente a cada um dos produtos (*array*) e a unidade de medida (*string*).

Já a estrutura *Materials* consiste numa lista de materiais. Esta possui um apontador para o *struct Materials materials* e dois inteiros, sendo estes o número de materiais presentes na lista (*nMaterials*) e o número máximo de materiais para a lista (*nMax*).

Listar Materiais – listMaterials

A função *listMaterials,* invocada selecionando a segunda opção do menu de gestão de materiais, recebe como parâmetro o *struct Materials* referente a lista e começa por validar se o número total de materiais existentes (*materials.nMaterials*) é maior que 0. Sendo esta condição verdadeira, é iniciado um *for loop* que executa a função printMaterials *n* vezes, sendo *n* o número de materiais. Já se a condição for falsa, é apresentado uma mensagem de erro no terminal informando da inexistência de materiais na lista.

Editar material – editMaterial

A função *editMaterial*, invocada ao selecionar a primeira opção do menu de gestão de materiais, recebe como parâmetros *o struct Materials* referente a lista e o *path* para o ficheiro contendo a lista de materiais (materials.bin). Esta função pede, primeiramente o *Id* do material a editar, depois verifica se o material existe usando a função *checkMaterial*. Se esse material existir, é pedido ao utilizador para inserir uma nova descrição, as quantidades desse material referente a cada um dos produtos e a unidade de medida. Caso o material não existir é apresentado uma mensagem de erro no terminal informando da inexistência de um material com esse Id na lista.

Gerar lista de produção

Estrutura de produção

A estrutura de produção é constituída por um *array* de inteiros com 23 posições referentes à quantidade de materiais que vão armazenar a quantidade do material associado a aquela posição.

generateProduction

A função *generateProduction*, invocada ao selecionar a quinta opção do menu de administração, recebe como parâmetros: o apontador para o *struct Orders* e outro para o *struct Orders*. Esta função é responsável de gerar, bem como exportar para um ficheiro à escolha do utilizador, uma lista de produção referente ao resumo dos componentes (materiais) a usar para satisfazer as encomendas de uma determinada semana ou data. Ela inicializa uma estrutura de produção e pede ao utilizador para inserir o intervalo de data a procurar. De seguida, invoca a função *checkOrdersDate* que compara a data de inicio e a data de fim fornecida com a data de entrega de cada a encomenda e caso a encomenda pertencer a esse intervalo, invoca-se a função *calculateMaterials*. Esta, por sua vez, calcula a quantidade de cada material para satisfazer essa

encomenda tendo em conta os produtos encomendados e adiciona a um total de cada material. Depois de serem verificadas todas as encomendas pertencentes ao intervalo, retorna-se para a função *generateProduction*, que apresenta a lista de produção para o intervalo de datas fornecido. Após apresentar a lista, pergunta ao utilizador se deseja exportar para um ficheiro e caso a resposta seja afirmativa, o utilizador digita o nome do ficheiro de texto para o qual quer exportar a lista e invoca-se a função responsável por essa tarefa – *saveProduction*.

3. Funcionalidades propostas

Estrutura de cidades

A estrutura de de cada cidade é composta por uma string com o nome da cidade e por um contador (*int*). Já a estrutura da lista de cidades é composta por um apontador para o *struct Cities* e por *nCities (int)*, que diz respeito à quantidade de cidades.

Cidades para entrega de encomendas em determinada data – ordersPerCityDate

Trata-se da função invocada ao escolher a primeira opção do menu de listagens. Ela recebe como parâmetros: o apontador para o *struct Orders* e o para o *struct Customers*. Primeiramente, inicializa-se duas estruturas de data e a estrutura de cidades.

Pede-se ao utilizador o intervalo de datas para procurar as encomendas e depois verificam-se todas as encomendas da lista. Caso a encomenda pertencer ao intervalo de datas, procura-se pelo cliente associado a essa encomenda na lista de clientes e adiciona-se uma unidade à cidade registada nos dados desse cliente na estrutura de cidades. Depois desse processo, é apresentado ao utilizador a lista de cidades para entrega de encomendas entre o intervalo determinado. Caso o cliente queira exportar essa lista, basta responder afirmativamente à pergunta e inserir o nome do ficheiro para o qual deseja exportar a lista.

Produtos ordenados por quantidade de encomendas – mostOrdersProducts

Esta função, invocada ao escolher a terceira opção do menu de listagens, recebe como para parâmetro a estrutura de encomendas e de produtos. Ela é responsável por apresentar a lista de produtos ordenados por quantidade de encomendas. Para isso, são cridos dois *arrays*, um para armazenar a quantidade de cada produto (*counter*) e outro para armazenar as posições por ordem de quantidade(*list*). Para cada encomenda, verificam-se os produtos encomendados e adiciona-se a quantidade respetiva a cada produto na posição associada ao produto no array counter. Depois de verificar todas as encomendas, os valores das posições do *counter* são comparados e ordenados no *array list*. Por fim, apresenta-se os produtos ordenados por ordem descendente de quantidade de unidades encomendadas.

Cidades com encomendas registadas – cityOrders

Esta função recebe como parâmetros apontadores para os structs *Orders* e *Customers*. Começa por criar o *struct Cities* que contém um *array* dinâmico do *struct City* e um inteiro *nCities* para contar o número de cidades. Aloca depois a memória para o *array cities* usando *malloc*. De seguida, itera pelo *array orders* dentro do *struct Orders.* Por cada encomenda, usa a função *checkCustomer* para encontrar o cliente associado com essa encomenda passando o *customerId* da encomenda e o struct *Customers*. Compara depois se a cidade desse cliente já existe no *struct Cities.* Se a cidade já existir no *struct*, é incrementado o contador dessa cidade. Já se a cidade não existir no *struct*, adiciona a cidade a esse mesmo *struct* e define o contador

como 1. Depois limpa o terminal e imprime a lista de cidades e a sua quantidade de encomendas. Esta verifica também se o utilizador pretende exportar esta lista para um ficheiro. Se o utilizador confirmar, a função lê um nome de ficheiro dado pelo utilizador, adiciona a extensão .txt, e chama a função *saveListOfCities* que guarda a lista de cidades para o ficheiro. Liberta também a memória alocada para o *array* das cidades usando *free*.

Cliente com mais encomendas – mostOrdersCustomer

Esta função recebe como parâmetros os structs Orders e um apontador para *o struct Customers customers*. Primeiro inicializa um *array counter* com o mesmo número de elementos que o número de clientes e define todos estes como 0. Em seguida, itera pelo conjunto de encomendas e, para cada encomenda, usa a função *checkCustomer* para encontrar a posição do cliente que fez a encomenda na lista de clientes. Se o cliente for encontrado, o elemento correspondente no *array counter* é incrementado por 1. Depois de percorrer todas as encomendas, esta itera o array *counter* para encontrar o cliente que tem o valor mais alto, ou seja, fez mais encomendas. De seguida, usa a posição do cliente na lista de clientes para exibir as informações deste (id, nome, nif, cidade) em conjuto com o número de pedidos que fez.

Encomendas ainda não entregues – notYetDelivered

Esta função recebe como parâmetros o *struct Orders* e um apontador para o *struct Products*. Começa por inicializar um variável *time_t (now)* e obter a data e horas atuais, usando depois a função *localtime()* para converter essa data num *struct* de tipo *tm* que irá ser usado para comparar com a data de cada encomenda. Depois, a função itera pelo *array orders* dentro do *struct Orders*. Por cada encomenda, a função compara a data da encomenda com a data de hoje. Se a data da encomenda é depois da data de hoje, imprime os detalhes dessa encomenda, incluindo o ID da encomenda, a data, o valor total e o nome e quantidade dos produtos encomendados.

Faturamento – moneyEarned

Esta função, invocada ao escolher a sexta opção do menu de listagens, recebe como parâmetros o *struct* de encomendas e a de produtos. Ela é responsável por calcular e apresentar a quantia faturada pela empresa. Ela percorre a lista de encomendas e calcula os preços dos produtos encomendados e envia para o *array* total na posição do respetivo produto. De seguida, organiza os produtos a serem apresentados por ordem de faturamento e calcula o total juntando as quantias de todos os produtos. Por fim, apresenta o total faturado pela empresa e as quantias associadas a cada produto, ordenado por ordem decrescente.

4. Estrutura analítica do projeto

O projeto começou a ser desenvolvido dias depois de ser anunciado. Com o grupo criado, começámos fazendo diversas pesquisas nomeadamente sobre apontadores (*Pointers*) e estruturas (*Structs*) pois, pelo fato de estarmos atrasados nas matérias em Fundamentos de Programação, não havíamos os conhecimentos necessários para desenvolver a grande maioria do projeto. Na primeira semana, passámos boa parte do tempo analisando o enunciado uma vez que a compreensão do problema é, na nossa visão, o principal fator de sucesso para o trabalho.

De seguida, fizemos uma subdivisão inicial do projeto em tarefas e sub-tarefas mais simples de forma a melhorar e orientar em relação ao planeamento temporal e definir os responsáveis das tarefas.

Subdivisão

Iniciou-se com um esboço do menu inicial do programa feito por todos os elementos do grupo, de maneira a entender ou imaginar possíveis fluxos de funcionamento do programa.

A parte de Gestão de clientes foi o primeiro conjunto de funcionalidades que foram desenvolvidas. Foi maioritariamente desenvolvida por David Santos, sofrendo algumas alterações com o decorrer do trabalho por todos os elementos do grupo no âmbito de satisfazer novas funcionalidades que iam sendo implementadas.

A gestão de produtos foi a segunda parte a ser desenvolvida. Depois do grupo passar por algumas dificuldades em encontrar a forma mais eficiente de gerir os produtos pré-definidos, decidiu-se que íamos gerir os produtos, inicialmente, sem associação com os materiais que os compõem (funcionalidade exigida para gerar listas de produção).

De seguida, ao encontrar uma solução eficiente que associasse os produtos aos materiais que os compõem, iniciou-se o desenvolvimento da parte de gestão de materiais por todos os elementos do grupo.

Terminada a primeira fase do projeto, passámos para o desenvolvimento das funcionalidades exigidas e propostas para encomendas. O principal encarregado foi David Santos embora, todos do grupo estiveram presentes no processo com ideias e participação na resolução de problemas.

Com a segunda fase pronta, o projeto sofreu algumas modificações no sentido de melhorar o funcionamento do programa, simplificar funções, organizar os vários menus e preparar para implementar as restantes funcionalidades exigidas.

As funcionalidades de produção foram as próximas a serem desenvolvidas pelo grupo, concluindo com as funcionalidades comuns exigidas. Com isso, passámos a desenvolver o projeto implementado as nossas próprias ideias e propondo novas funcionalidades úteis para o contexto do projeto.

Desta forma, o projeto foi sofrendo várias melhorias tanto a nível de código como de aparência. Enquanto isso, o grupo desenvolveu funcionalidades de para gestão de listagens e relatório de acordo com interesses da empresa tendo em conta os dados armazenados e utilizados pelo programa.

Finalmente, foram feitos vários testes com intuito de testar o funcionamento do programa como um todo e foram feitas as configurações e comentários úteis à documentação doxygen.

Durante a maioria do processo, trabalhou-se no relatório, o que não só, melhorou na compreensão do problema, como despertou novas ideias a implementar.

5. Funcionalidades implementadas

Menus

Menu Inicial (login) - useMenu

A função *useMenu* trata-se da primeira interação com o utilizador e é uma espécie de tela de login. Ele é responsável por solicitar pelo ID (identificação) do utilizador e decidir qual será o menu (função) para o qual este será enviado. Inserindo:

- 0(zero) - o programa assume que o utilizador em questão não é registado sendo assim, é invocado a função de criação de cliente – *createCustomers*;

- 1001 - Definido como perfil de Administrador, é invocado o menu de administração – *useAdminMenu*, após validar pela password deste perfil usando a função *getPassword*. Está função pede e valida a password do Administrador. O utilizador possui 3 tentativas. Caso acertar, como mencionado anteriormente, é invocado o menu de administração, caso contrário, continua na função inicial *useMenu;*
- 1002 - O programa termina após libertar a memória;
- Qualquer outro ID que pertença ao intervalo válido - é invocada a função de verificação do cliente – *checkCustomer*. Caso o cliente existir, é invocado o menu de clientes – *useCustomerMenu*, caso contrário, continua na função inicial *useMenu.*

Menu de cliente – useCustomerMenu

O menu de cliente apresenta 5 opções nomeadamente: editar cliente, registar encomenda, ver as minhas encomendadas, deletar cliente e voltar para menu inicial. Selecionando:

- Editar cliente – é invocada a função *editCustomer* para modificar os dados relativos ao respetivo cliente;
- Registar encomenda – é invocada a função *createOrders* para criar um novo registo de encomenda;
- Ver as minhas encomendas – é invocada a função *viewMyOrders* que faz print de todas as encomendas já feitas pelo cliente em questão;
- Deletar cliente – é invocada a função *deleteCustomer* para eliminar o respetivo cliente caso este não possuir encomendas efetuadas e não entregues, ou mudar o estado para inativo caso possua;
- Voltar – retorna para o menu inicial, ou seja, dá *logout*.

Esta função vai ser executada enquanto não for escolhida a opção voltar.

Menu Administrador – useAdminMenu

O menu de administração é onde se encontra a esmagadora maioria das funcionalidades do programa. Trata-se de um menu protegido por *password*

Gestão de Clientes

É invocada a função *manageCustomers* na qual encontra-se o menu para as funcionalidades requeridas respetivas à gestão de clientes nomeadamente: editar cliente, …;

O menu de cliente apresenta 7 opções de submenus nomeadamente: gestão de clientes, gestão de produtos, gestão de encomendas, gestão de materiais, gestão de produção, listagens e voltar para menu inicial. Selecionando:

Gestão de Produtos

É invocada a função *manageProducts* na qual se encontra o menu para as funcionalidades requeridas respetivas à gestão dos produtos;

Gestão de Encomendas

É invocada a função *manageOrders* na qual encontra-se o menu para as funcionalidades necessárias respetivas à gestão das encomendas;

Gestão de Produção

É invocada a função *generateProduction* na qual encontra-se as funcionalidades requeridas respetivas à gestão de produção;

Gestão de Listagens

É invocada a função *manageLists* na qual encontra-se o menu com opções das diferentes possíveis listagens/relatórios;

Voltar

Retorna para o menu de login.

Gestão de ficheiros

Inserção de dados – insert(x)

As funções *insertCustomer, insertProduct e insertOrder,* recebem como parâmetros o seu struct respetivo e uma string que identifica o ficheiro a ser aberto e modificado, sendo este definido nos headers. Estas funções começam por tentar abrir o ficheiro respetivo a cada conjunto de dados, verificando se este existe, se este não existir, o programa é terminado, caso contrário, o programa escreve os dados provenientes de outras funções, guardados nos respetivos structs, no dito ficheiro. Após este processo o ficheiro é fechado.

Remoção de dados – remove(x)

As funções *removeCustomer*, *removeProduct e removeOrder,* recebem como parâmetros o seu struct respetivo e uma string que identifica o ficheiro a ser aberto e modificado, sendo este definido nos headers. Estas funções começam por tentar abrir e reescrever o ficheiro respetivo a cada conjunto de dados, criando um novo caso este não exista. É verificado de seguida se o ficheiro existe, se este não existir, o programa é terminado, caso contrário,

Edição de dados – update(x)

As funções *updateCustomer, updateProduct, updateMaterial e updateOrder,* recebem como parâmetros o seu struct respetivo e uma string que identifica o ficheiro a ser aberto, a posição do produto no ficheiro (pos) e modificado, sendo este definido nos headers. Estas funções começam por tentar abrir o ficheiro respetivo a cada conjunto de dados, verificando se este existe, se este não existir, o programa é terminado, caso contrário, procuram no ficheiro a linha guardada em *pos* e reescrevem essa mesma linha com a nova informação proveniente duma das outras funções.

Ver as minhas encomendas - viewMyOrders

A funcionalidade de ver as minhas encomendas é de responsabilidade da função viewMyOrders, invocada selecionando a terceira opção do menu de cliente (logado). Trata-se de uma função que recebe como parâmetros: o apontador para a estrutura de encomendas, outro para a de produtos e o Id do cliente. A função percorre a lista de encomendas a procura de encomendas registadas com o Id desse cliente. Caso uma encomenda possuir o id desse cliente, ela é apresentada.

6. Conclusão

Ao longo do projeto foi desenvolvida uma aplicação com o principal objetivo de satisfazer as exigências e necessidades da empresa, bem como cumprir com todos os objetivos estabelecidos inicialmente.

Pesquisas e aprendizado autónomo foram cruciais para o sucesso do projeto permitindo ao grupo melhorar as suas abilidades de resolução de problemas, possibilitando a melhoria de várias funcionalidades da aplicação.

A completa compreensão do projeto e do contexto da sua implementação possibilitou não só o desenvolvimento de várias funcionalidades adicionais, como também, a criação de listagens e relatórios úteis para a empresa.