ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Projeto de Laboratório de Programação

Licenciatura em Engenharia Informática
2021/2022

Grupo 305
Luís Carlos Mendes de Oliveira – 8190370

1. Introdução

O projeto desenvolvido consiste num programa para a gestão de stocks de matéria-prima para a satisfação das ordens de produção, bem como a geração de notas de encomenda para a matéria-prima em falta.

Através da utilização deste programa, o utilizador consegue importar o stock existente de um ficheiro para o programa, bem como as necessidades das ordens de produção. Após estas duas etapas é gerada a encomenda. A encomenda tem que ter em conta o stock mínimo a ter de cada matéria-prima bem como as quantidades que o Fornecedor vende de cada matéria-prima.

O programa também permite listar todas as encomendas de matérias-primas realizadas

No decorrer do projeto foram tomadas algumas decisões para resoluções de problemas tais como: os ficheiros fornecidos foram convertidos para o formato ".txt"

Tendo em conta que este projeto diz respeito à gestão de stocks e encomendas de uma empresa da área de produção de calçado e tendo por base o conhecimento que o grupo tem da área em questão, foi assumido que o stock máximo que é possível inserir é de 10000000 para cada matérias-primas. Para o grupo este valor é um valor seguro pois, tendo em conta os consumos de matérias-primas necessárias para a produção de um artigo de calçado.

A titulo de exemplo, uma bota 42 tem de consumo entre 4 a 5 pés^2 de pele (couro)/tecido, o que equivale a 0.46m^2 , ou seja, 10000000 m^2 seriam o suficiente para 21739130 pares de botas. Tendo em conta em 2020 Portugal exportou 61 milhões de pares de calçado [1] e a maior parte da produção nacional destina-se a exportação, consideramos que este valor é um valor seguro para ter como máximo de stock de matérias primas.

2. Funcionalidades requeridas

2.1 Gestão de Produção

Neste programa é possível importar ordens de produção de um ficheiro de texto de forma a calcular as necessidades totais de matéria-prima.

O grupo utilizou uma estrutura "Necessidades" para armazenar as quantidades necessárias para produzir a(s) nota(s) de produção.

```
typedef struct {
   int necessidadesCouro;
   int necessidadesTecido;
   int necessidadesBorracha;
   int necessidadesCordoes;
   int necessidadesPalmilhas;
} Necessidades;
```

Esta estrutura contém 5 variáveis do tipo *int* que armazenam as necessidades de cada uma das 5 diferentes matérias-primas.

A leitura das quantidades que se encontram no ficheiro com a nota de produção é feita através da função “carregarNecessidades”.

Esta função abre o ficheiro com a ordem de produção para leitura. Depois é armazenado o número de linhas que o ficheiro tem na variável “numeroLinhas” recorrendo à função “nrLinhasFicheiro”. É armazenado o número de linhas de produção em “nrLinhasOP” e passa o texto restante da linha à frente[2]. As linhas de produção são lidas para uma variável apenas para fazer a leitura seletiva do texto enquanto a variável “i” é melhor que “nrLinhasOP+8”. Utilizou-se o *fscan* para a leitura de *string*, lendo até encontrar o caracter espaço. Como o ficheiro contém espaços, surge “+8” e “+6” para contornarem esta situação. Finalmente é alcançada a parte que contem as necessidades. Aqui é lido o código e a quantidade, e consoante o código, a quantidade é adicionada à variável da correspondente da estrutura “Necessidades.

```
void carregarNecessidades(Necessidades *necessidades, char *nomeFicheiro) {

   int i = 0, op, nrLinhasOP, materiaPrima,quantidade;
   char stringAux [50]; //variavel "fake" para amazenar linhas desnecessarias

   necessidades->necessidadesCouro = 0;
   necessidades->necessidadesTecido = 0;
   necessidades->necessidadesBorracha = 0;
   necessidades->necessidadesCordoes = 0;
   necessidades->necessidadesPalmilhas = 0;

   FILE *fp = fopen(nomeFicheiro, "r");

   int numeroLinhas = nrLinhasFicheiro(nomeFicheiro);
   if (fp != NULL) {
      if (numeroLinhas > 0) {
         fscanf(fp, "%d %*[^\n]", &nrLinhasOP);//non-capturing scan[2]

         while(i<nrLinhasOP+8){ //8= valor para contornar espaços existentes no ficheiro
            fscanf(fp, "%s", stringAux);
            i++;
         }
         while (i < numeroLinhas+6){ //6 = valor para contornar espaços existentes no ficheiro
            fscanf(fp, "%d,%d,%d", &op, &materiaPrima, &quantidade);

            if (materiaPrima == 1) {
               necessidades->necessidadesCouro += quantidade;
            } else if (materiaPrima == 2) {
               necessidades->necessidadesTecido += quantidade;
            } else if (materiaPrima == 3) {
               necessidades->necessidadesBorracha += quantidade;
            } else if (materiaPrima == 4) {
               necessidades->necessidadesCordoes += quantidade;
            } else if (materiaPrima == 5) {
               necessidades->necessidadesPalmilhas += quantidade;
            }
            i++;
         }
         printf("1 - Couro - %d\n", necessidades->necessidadesCouro);
         printf("2 - Tecido - %d\n", necessidades->necessidadesTecido);
         printf("3 - Borracha - %d\n", necessidades->necessidadesBorracha);
         printf("4 - Cordoes - %d\n", necessidades->necessidadesCordoes);
         printf("5 - Palmilhas - %d\n", necessidades->necessidadesPalmilhas);
      } else {
```

```
            puts("Error");
        }
    } else {
        puts(MSG_ERRO_LER_FICHEIRO);
    }
    fclose(fp);
}
```

Função “nrLinhasFicheiro”:

```
int nrLinhasFicheiro(char *filename){
    char ch;
    FILE *file;
    file = fopen(filename, "r");
    int linhas = 0;

    if (file != NULL) {
        for (ch = getc(file); ch != EOF; ch = getc(file))
        if (ch == '\n')
            linhas = linhas + 1;
    }
    fclose(file);

    return linhas - 1;
}
```

2.2 Gestão de stock de matéria-prima

A estrutura “Stock” armazena a quantidade em stock de cada matéria-prima bem como o valor de stock mínimo estipulado.

```
typedef struct {
    int couroStock;
    int minStockCouro;
    int tecidoStock;
    int minStockTecido;
    int borrachaStock;
    int minStockBorracha;
    int cordoesStock;
    int minStockCordoes;
    int palmilhasStock;
    int minStockPalmilhas;
} Stock;
```

Através da função “criarStock” é possível ler da consola os para cada variável. O valor do mínimo de Stock não pode ser inferior ao valor em stock.

```
3.  void criarStock(Stock *stock){
4.
5.      puts("===== Inserir dados de stock: =====\n");
6.      imprimeLinha(36);
7.      stock->couroStock = obterInt(0,MAX_STOCK, "Quantidade de couro em stock (metros): ");
8.      stock->minStockCouro = obterInt(0,stock->couroStock, "Quantidade minima de stock de couro
    (metros): ");
9.      stock->tecidoStock = obterInt(0,MAX_STOCK, "Quantidade de tecido em stock (metros): ");
10.     stock->minStockTecido = obterInt(0,stock->tecidoStock, "Quantidade minima de stock de tecido
    (metros): ");
```

```
11.     stock->borrachaStock = obterInt(0,MAX_STOCK, "Quantidade de borracha em stock (Kg): ");
12.     stock->minStockBorracha = obterInt(0,stock->borrachaStock, "Quantidade minima de stock de
        borracha (Kg): ");
13.     stock->cordoesStock = obterInt(0,MAX_STOCK, "Quantidade de cordoes em stock (metros): ");
14.     stock->minStockCordoes = obterInt(0,stock->cordoesStock, "Quantidade minima de stock de
        cordoes (metros): ");
15.     stock->palmilhasStock = obterInt(0,MAX_STOCK, "Quantidade de palmilhas em stock (pares):
        ");
16.     stock->minStockPalmilhas = obterInt(0,stock->palmilhasStock, "Quantidade minima de stock de
        palmilhas (pares): ");
17.
18. }
```

A função "carregarStock" abre o "tabela_stocks.txt" com os valores de stock e quantidade mínimas por matéria-prima para leitura. À semelhança de função de leitura de ficheiro descrita anteriormente, esta também faz uso da "nrLinhasFicheiro" para saber o número de linhas que o ficheiro tem. Através de um ciclo *while,* após a leitura da primeira linha que só contem *strings,* o ficheiro é percorrido linha a linha, sendo armazenado o valor do código, mínimo de stock e quantidade de stock nas variáveis "código", "mínimo" , "quantidade" respetivamente.

Através do valor do código, é filtrado recorrendo a *"if's"* as variáveis da estrutura stock correta. Exemplo, se código é 1, então refere-se a couro.

```
void carregarStock(Stock *stock, char *nomeFicheiro){
   int i = 0;
   int codigo; //variavel para armazenar o codigo
   char stringAux [40]; //variavel "fake" para amazenar nome
   int minimo, quantidade;

   FILE *fp = fopen(nomeFicheiro, "r");

   int numeroLinhas = nrLinhasFicheiro(nomeFicheiro);

   if (fp != NULL){

      if(numeroLinhas >0){
         fscanf(fp, "%s", stringAux);//skip first line

         while(i< numeroLinhas){
            fscanf(fp, "%d, %*[^,],%d,%d", &codigo, &minimo, &quantidade);

            if(codigo==1){
               stock->minStockCouro = minimo;
               stock->couroStock = quantidade;
            }else if(codigo==2){
               stock->minStockTecido = minimo;
               stock->tecidoStock = quantidade;
            }else if(codigo==3){
               stock->minStockBorracha = minimo;
               stock->borrachaStock = quantidade;
            }else if(codigo==4){
               stock->minStockCordoes = minimo;
               stock->cordoesStock = quantidade;
            }else if(codigo==5){
               stock->minStockPalmilhas = minimo;
               stock->palmilhasStock = quantidade;
            }
```

```
            i++;
        }

    }else{
        puts("Error");
    }
    puts("\n!Stock carregado com sucesso do ficheiro!\n\n");
  }else{
    puts(MSG_ERRO_LER_FICHEIRO);
  }
  fclose(fp);
}
```

A função “mostrarStock” imprime na consola as informações do stock, permitindo ao utilizador visualizar o stock existente.

```
void mostrarStock(Stock stock){
  puts("==============  Stock : ==============\n");
  puts("Codigo | Produto | Stock Minimo | Stock Atual\n");
  printf("1\tCouro\t\t%d\t\t%d\n", stock.minStockCouro, stock.couroStock);
  printf("2\tTecido\t\t%d\t\t%d\n", stock.minStockTecido, stock.tecidoStock);
  printf("3\tBorracha\t%d\t\t%d\n", stock.minStockBorracha, stock.borrachaStock);
  printf("4\tCordões\t\t%d\t\t%d\n", stock.minStockCordoes, stock.cordoesStock);
  printf("5\tPalmilhas\t%d\t\t%d\n", stock.minStockPalmilhas, stock.palmilhasStock);
}
```

É possível editar o stock existente, caso a situação assim o justifique, através da função “editarStock”. O utilizador seleciona qual(s) matéria(s)-prima(s) pretende editar, ficando a alteração guardada na memória central do computador.

```
void editarStock(Stock *stock){
  int op = -1;

  do{
    puts("1 - Stock actual de couro");
    puts("2 - Stock minimo de couro a ter");
    puts("3 - Stock actual de tecido");
    puts("4 - Stock minimo de tecido a ter");
    puts("5 - Stock actual de borracha");
    puts("6 - Stock minimo de borracha a ter");
    puts("7 - Stock actual de cordões");
    puts("8 - Stock minimo de cordões a ter");
    puts("9 - Stock actual de palmilhas");
    puts("10 - Stock minimo de palmilhas a ter");
    puts("0 - Sair");
    op = obterInt(0, 10, "=== Qual valor pretende alterar: ===\n");

    if(op==1){
      stock->couroStock = obterInt(stock->minStockCouro, MAX_STOCK,"Novo stock couro: ");
    }if(op == 2){
      stock->minStockCouro = obterInt(0,stock->couroStock, "Novo stock minimo couro: ");
    }if(op == 3){
      stock->tecidoStock = obterInt(stock->minStockTecido, MAX_STOCK,"Novo stock tecido: ");
    }if(op == 4){
      stock->minStockTecido = obterInt(0,stock->tecidoStock , "Novo stock minimo tecido: ");
    }if(op == 5){
```

```
            stock->cordoesStock = obterInt(stock->minStockCordoes, MAX_STOCK,"Novo stock cordoes:
");
        }if(op == 6){
            stock->minStockCordoes = obterInt(0,stock->cordoesStock  , "Novo stock minimo cordoes: ");
        }if(op == 7){
            stock->borrachaStock = obterInt(stock->minStockBorracha, MAX_STOCK,"Novo stock
borracha: ");
        }if(op == 8){
            stock->minStockBorracha = obterInt(0,stock->borrachaStock  , "Novo stock minimo borracha: ");
        }if(op == 9){
            stock->palmilhasStock = obterInt(stock->minStockPalmilhas, MAX_STOCK,"Novo stock
palmilhas : ");
        }if(op == 10){
            stock->minStockPalmilhas = obterInt(0,stock->palmilhasStock  , "Novo stock minimo palmilhas:
");
        }
    }while(op != 0);
}
```

De forma a garantir a persistência de dados, é possível guardar o stock atual em ficheiro recorrendo a "guardarStock" que guarda no ficheiro "tabela_stocks" o stock atual a pedido do utilizador.

A estrutura "MinimoEnc" foi utilizada para armazenar as quantidades mínimas que o Fornecedor permite encomendar.

```
 typedef struct {
    int couro;
    int tecido;
    int borracha;
    int cordoes;
    int palmilhas;
} MinimoEnc;
```

Sendo as quantidades do enunciado definidas através da função "criarMinimoEncomenda" :

```
void criarMinimoEncomenda(MinimoEnc *minEnc) {
    minEnc->couro = 7; //1 rolo de 7m
    minEnc->borracha = 5; //1 saco de 5kg
    minEnc->cordoes = 10; //rolo 10 metros de cordões
    minEnc->palmilhas = 50; // saco de 50 palmilhas
    minEnc->tecido = 0; //sem minimo
}
```

Estes valores são possíveis de alterar recorrendo à função "editarMinimoEncomenda":

```
void editarMinimoEncomenda(MinimoEnc *minEnc) {
    puts("======= Editar Quantidade Minima Possivel de Encomendar =======");
    minEnc->couro = obterInt(0, 10000, "Couro - minimo a encomendar: ");
    minEnc->borracha = obterInt(0, 10000, "Borracha - minimo a encomendar: ");
    minEnc->cordoes = obterInt(0, 10000, "Cordões - minimo a encomendar: ");
    minEnc->palmilhas = obterInt(0, 10000, "Palimilhas - minimo a encomendar: ");
    minEnc->tecido = obterInt(0, 10000, "Tecido - minimo a encomendar: ");
}
```

2.3 Registo de ordens de compra (encomendas)

Após o stock ser preenchido ou carregado de ficheiro e as necessidades da nota de produção, chegamos à etapa de criar a encomenda de matéria-prima que será enviada ao fornecedor nas quantidades fornecidas por este.

A estrutura "Encomenda" representa a ordem de compra:

```
typedef struct {
   int codigoEncomenda; //Unico
   int encomendaCouro;
   int encomendaTecido;
   int encomendaBorracha;
   int encomendaCordoes;
   int encomendaPalmilhas;
} Encomenda;
```

Nela as quantidades armazenadas estão de acordo com as quantidades do fornecedor. Exemplo: o fornecedor vende o couro em rolos de 7 metros, então o que estará na encomenda será a quantidade de rolos de 7 metros de couro a encomendar e o mesmo acontece para as outras matérias-primas. Além disso, contém a variável "codigoEncomenda" que é o número da encomenda.

A encomenda é criada recorrendo a "criarEncomenda". Tal como referido acima, é necessário que o stock e as necessidades já estejam atribuídos. Nesta função será calculado a quantidade a encomendar. O valor da variável "calculaQuantidade" irá ditar o comportamento da encomenda da matéria-prima em questão. É verificado se há quantidade suficiente em stock, sem comprometer a quantidade mínima de stock, se há stock, mas mesmo assim é preciso encomendar mais pois o stock ficará abaixo da quantidade mínima ou se não é mesmo preciso encomenda pois a quantidade em stock é insuficiente, mesmo contando com o mínimo de stock. No caso de ser preciso encomendar, é verificado se a matéria-prima em questão tem uma unidade/quantidade exigida pelo fornecedor. Não tendo, o valor anteriormente calculado é o valor da encomenda. Havendo, é preciso garantir que o valor calculado satisfaz as exigências do fornecedor. Supondo que são necessários 10 metros de couro, o fornecedor vende rolos de 7, a encomenda terá que ser de 2 rolos pois o fornecedor não vende "meios" rolos. Esta quantidade excedente da encomenda irá sobrar e passar a contar para stock.

```
void criarEncomenda(Encomenda *enc, MinimoEnc minEnc, Stock *st, Necessidades *nec) {

   int calculaQuantidade, temp;

   calculaQuantidade = nec->necessidadesCouro - st->couroStock + st->minStockCouro;

   if (calculaQuantidade > 0) {
      if (minEnc.couro != 0) {
         if (calculaQuantidade % minEnc.couro != 0) {
            enc->encomendaCouro = calculaQuantidade / minEnc.couro + 1;
         } else {
            enc->encomendaCouro = calculaQuantidade / minEnc.couro;
         }
```

```
            temp = enc->encomendaCouro * minEnc.couro;
            temp = temp - calculaQuantidade;
            st->couroStock = temp + st->minStockCouro;
        } else {
            enc->encomendaCouro = calculaQuantidade;
            st->couroStock = st->minStockCouro;
        }
    } else if (calculaQuantidade < 0) {
        if ((-1) * calculaQuantidade > st->minStockCouro) {
            enc->encomendaCouro = 0;
            st->couroStock -= nec->necessidadesCouro;
        } else {
            enc->encomendaCouro = 0;
            st->couroStock += calculaQuantidade;
        }
    } else {
        enc->encomendaCouro = 0;
        st->couroStock -= nec->necessidadesCouro;
    }


    calculaQuantidade = nec->necessidadesTecido - st->tecidoStock + st->minStockTecido;
    …
    // CÒDIGO REPETE-SE, SÓ MUDAM AS VARIAVEIS DAS MATERIAS-PRIMAS

    imprimeEncomenda(*enc);
}
```

A encomenda criada pode então ser guardada no ficheiro "Encomenda.txt" através de "guardarEncomenda". Notar que apenas são guardadas as matérias-primas que sejam realmente precisas encomendar. O número da encomenda é também guardado.

```
void guardarEncomenda(Encomenda *enc){
    FILE *fp = fopen("Encomenda.txt", "w");
    if (fp != NULL){
        fprintf(fp,"==============  Encomenda nr: %d : ============\n", enc->codigoEncomenda
);
        fprintf(fp,"Codigo | Produto | Quantidade\n");
        if(enc->encomendaCouro>0){
            fprintf(fp,"1\tCouro\t\t%d rolos\n", enc->encomendaCouro);
        }
        if(enc->encomendaTecido>0){
            fprintf(fp,"2\tTecido\t\t%d metros\n", enc->encomendaTecido);
        }
        if(enc->encomendaBorracha>0){
            fprintf(fp,"3\tBorracha\t%d sacos\n", enc->encomendaBorracha);
        }
        if(enc->encomendaCordoes>0){
            fprintf(fp,"4\tCordões\t\t%d metros\n", enc->encomendaCordoes);
        }
        if(enc->encomendaPalmilhas>0){
            fprintf(fp,"5\tPalmilhas\t%d sacos\n", enc->encomendaPalmilhas);
        }
    }else{
        puts("ERRO NA ESCRITA DO FICHEIRO");
    }
}
```

Para armazenar as várias encomendas que sejam criadas, o grupo fez uso da A estrutura “ListaEncomendas” que contém um apontador para a estrutura “Encomenda” que irá servir para alocar memória. Contém também dois valores inteiros, um para o tamanho da lista e outro para o número de encomendas na lista.

```
typedef struct {
   int nrEncomendas;
   int tamanho;
   Encomenda *listaEncomendas;
} ListaEncomendas;
```

As encomendas são adicionadas à lista de encomendas através da função “adicionarEncomenda”:

```
int adicionarEncomenda(ListaEncomendas *lista, Stock *st, Necessidades *nec, MinimoEnc minEnc){
   Encomenda enc;
   enc.codigoEncomenda = lista->nrEncomendas + 1;
   criarEncomenda(&enc, minEnc, st, nec);
   //verifica se  codigo da encomenda já existe
   for (int i = 0; i < lista->nrEncomendas; i++) {
      if (lista->listaEncomendas[i].codigoEncomenda == enc.codigoEncomenda){
         return 0;
      }
   }
   //lista cheia
   if (lista->nrEncomendas == lista->tamanho) {
      expandirMemoria(lista);
   }

   // adiciona e incrimenta o nr de funcionários
   lista->listaEncomendas[lista->nrEncomendas] = enc;
   lista->nrEncomendas++;
   return 1;
}
```

As encomendas presentes na lista de encomendas podem ser exportadas para o ficheiro “ListadeEncomendas.txt” através da função:

```
void guardarListaEncomendas(ListaEncomendas *lista) {
   if (lista->nrEncomendas == 0) {
      puts("\nSem encomendas!\n");
   } else {
      for (int i = 0; i < lista->nrEncomendas; i++) {
         guardarEncomendas(&lista->listaEncomendas[i]);
         puts("");
      }
   }
}

void guardarEncomendas(Encomenda *enc){

   FILE *fp = fopen("ListadeEncomendas.txt", "a+");
   if (fp != NULL){
      fprintf(fp,"==============  Encomenda nr: %d : ==============\n", enc->codigoEncomenda
);
```

```
        fprintf(fp,"Codigo | Produto | Quantidade\n");
        if(enc->encomendaCouro>0){
            fprintf(fp,"1\tCouro\t\t%d rolos\n", enc->encomendaCouro);
        }
        if(enc->encomendaTecido>0){
            fprintf(fp,"2\tTecido\t\t%d metros\n", enc->encomendaTecido);
        }
        if(enc->encomendaBorracha>0){
            fprintf(fp,"3\tBorracha\t%d sacos\n", enc->encomendaBorracha);
        }
        if(enc->encomendaCordoes>0){
            fprintf(fp,"4\tCordões\t\t%d metros\n", enc->encomendaCordoes);
        }
        if(enc->encomendaPalmilhas>0){
            fprintf(fp,"5\tPalmilhas\t%d sacos\n", enc->encomendaPalmilhas);
        }
    }else{
        puts("ERRO NA ESCRITA DO FICHEIRO");
    }
}
```

O utilizador também pode consultar a encomenda que pretender, introduzindo o número da mesma:

```
void mostraEncomenda(ListaEncomendas *lista, int nrEnc) {
    Encomenda *enc;
    enc = obterEncomenda(lista, nrEnc);
    if (enc != NULL) {
        imprimeEncomenda(*enc);
    } else {
        puts("Encomenda não encontrada");
    }
}

Encomenda* obterEncomenda(ListaEncomendas *lista, int nrEnc) {
    if (nrEnc <= 0 && nrEnc > lista->nrEncomendas) {
        puts("Número de encomenda Inválido");
        return NULL;
    } else {
        return &lista->listaEncomendas[nrEnc-1];
    }
}

void imprimeEncomenda(Encomenda enc){
    printf("==============  Encomenda nr: %d : ==============\n", enc.codigoEncomenda);
    puts("Codigo | Produto | Quantidade\n");
    printf("1\tCouro\t\t%d rolos\n", enc.encomendaCouro);
    printf("2\tTecido\t\t%d metros\n", enc.encomendaTecido);
    printf("3\tBorracha\t%d sacos\n", enc.encomendaBorracha);
    printf("4\tCordões\t\t%d metros\n", enc.encomendaCordoes);
    printf("5\tPalmilhas\t%d sacos\n\n\n", enc.encomendaPalmilhas);
}
```

2.5 Memória dinâmica

Para adicionar um novo elemento à lista “ListaEncomendas” é verificado o tamanho da memória existente bem como a quantidade de informação armazenada

na memória. Se o limite de memória é alcançado, é o tamanho é realocado para o dobro.

```
void expandirMemoria(ListaEncomendas *lista){
   Encomenda *enc = (Encomenda*) realloc(lista->listaEncomendas, sizeof(Encomenda) * (lista-
>tamanho * 2));
   lista->listaEncomendas= enc;
   lista->tamanho *= 2;
}
```

3. Funcionalidades propostas

O grupo propôs 3 listagens.

A primeira "mediaStock" permite auxiliar a produção na tomada de decisões, em caso de falha de fornecimento, permitindo saber quanto é possível gastar do stock sem comprometer o stock mínimo de cada matéria-prima.

```
void mediaStock( Stock stock){
   int total = stock.borrachaStock + stock.cordoesStock + stock.palmilhasStock + stock.tecidoStock +
stock.couroStock;
   printf("A media de stock de cada matéria prima é  %.2f\n", (total /(float) 5));

   printf("Podemos gastar [ %d ] de couro sem afetar o minimo de stock \n",stock.couroStock-
stock.minStockCouro );
   printf("Podemos gastar [ %d ] de Tecido sem afetar o minimo de stock \n", stock.tecidoStock-
stock.minStockTecido);
   printf("Podemos gastar [ %d ] de Borracha sem afetar o minimo de stock \n", stock.borrachaStock-
stock.minStockBorracha);
   printf("Podemos gastar [ %d ] de cordões sem afetar o minimo de stock \n", stock.cordoesStock-
stock.minStockCordoes);
   printf("Podemos gastar [ %d ] de Palmilhas sem afetar o minimo de stock \n", stock.palmilhasStock-
stock.minStockPalmilhas);
}
```

A segunda listagem, "listarEncomendas" permite saber quantas encomendas foram feitas. Esta informação é importante para controlo dos movimentos de stock.

```
void listarEncomendas(ListaEncomendas *lista) {
   if (lista->nrEncomendas == 0) {
      puts("\nSem encomendas!\n");
   } else {
      for (int i = 0; i < lista->nrEncomendas; i++) {
         puts("");
         printf("Encomenda nª %d\n", lista->listaEncomendas[i].codigoEncomenda);
         imprimeEncomenda(lista->listaEncomendas[i]);
         puts("");
      }
   }
}
```

Por fim, "mediaEncomendas", listagem que, tendo em conta todas as encomendas feitas, determina a quantidade média de encomenda de cada matéria-prima. O grupo considera importante esta listagem pois permite por exemplo à

empresa fazer cálculos de aproveitamento de matérias-primas tendo em conta as produções, verificar se os valores mínimos de stock estão corretos.

```
void mediaEncomendas(ListaEncomendas *lista){
   int encomendaCouro = 0;
   int encomendaTecido = 0;
   int encomendaBorracha = 0;
   int encomendaCordoes = 0;
   int encomendaPalmilhas = 0;

   for (int i = 0; i < lista->nrEncomendas; i++) {
      encomendaCouro += lista->listaEncomendas[i].encomendaCouro;
      encomendaTecido += lista->listaEncomendas[i].encomendaTecido;
      encomendaBorracha += lista->listaEncomendas[i].encomendaBorracha;
      encomendaCordoes += lista->listaEncomendas[i].encomendaCordoes;
      encomendaPalmilhas += lista->listaEncomendas[i].encomendaPalmilhas;
   }
   printf("Cada encomenda em média possui de Couro %.1f \n", (encomendaCouro /(float)lista-
>nrEncomendas));
   printf("Cada encomenda em média possui de Tecido %.1f \n", (encomendaTecido /(float)lista-
>nrEncomendas));
   printf("Cada encomenda em média possui de Borracha %.1f \n", (encomendaBorracha /(float)lista-
>nrEncomendas));
   printf("Cada encomenda em média possui de cordões %.1f \n", (encomendaCordoes /(float)lista-
>nrEncomendas));
   printf("Cada encomenda em média possui de Palmilhas %.1f \n\n\n", (encomendaPalmilhas
/(float)lista->nrEncomendas));
}
```

4. Estrutura analítica do projeto

O elemento do grupo tentou aproveitar ao máximo a sua agenda e os conhecimentos na área da indústria de calçado para a elaboração deste trabalho.

A elaboração seguiu a seguinte ordem de criação: Stock, Necessidades, Encomendas, ListaEncomendas e por fim o menu presente em Main.

5. Funcionalidades implementadas

Gestão de Produção - Todas as funcionalidades propostas para a gestão de Produção foram implementadas.

Gestão de stock de matéria-prima - Todas as funcionalidades propostas para a gestão de stock de matéria-prima foram implementadas.

Registo de ordens de compra - Todas as funcionalidades propostas para Registo de ordens de compra foram implementadas.

Persistência de dados - a aplicação permite guardar/carregar dados em/de ficheiro, permitindo persisti-los ao longo de diferentes utilizações, de forma a possibilitar a leitura e gravação a qualquer momento através da respetiva escolha no menu de opções

Memoria Dinâmica - Foi implementada

Listagens de dados – Foram propostas 3 listagens

6. Bibliografia

1 – https://www.apiccaps.pt/library/media_uploads/jornalnoticiasapiccaps-285.pdf

2 - https://stackoverflow.com/questions/30065675/what-does-scanf-nc-mean

7. Conclusão

Na realização deste projeto foram identificar falhas de conhecimento em Linguagem C. Este projeto surge da tentativa de as colmatar. A utilização de documentação fornecida nesta e noutras UC's, bem como documentação online, foi essencial.