Alinea A

Os casos de teste visíveis são também execuções de exemplo.

Para esta alínea, basta ler o ficheiro e apresentá-lo. Tinham por exemplo a AF more.c. Aqui a estrutura de dados mais simples e que serviria para as restantes alíneas seria a lista de strings:

```
typedef struct SProduto
{
        char *nome;
        struct SProduto *seguinte;
} TProduto;
```

Uma função para adicionar uma linha, outra para libertar a lista, e uma outra função para carregar o ficheiro.

Para mostrar os produtos (que poderia ser outra função), bastariam duas ou três linhas:

```
        printf("Produtos:");
        while (lista != NULL) {
                printf("\n- %s;", lista->nome);
                lista = lista->seguinte;
        }
```

Naturalmente que poderiam utilizar também vetores, mas esta solução é mais simples.

Alinea B

Os casos de teste visíveis são considerados também execuções de exemplo.

Aqui poderia ser tomada a decisão de fazer a estrutura de dados apenas para responder a esta alínea. Assim bastaria contar o número de linhas entre produtos. Mas naturalmente que esta solução não serve para solicitações futuras. Apenas uma estrutura de dados que leia os dados sem perdas, é que tem hipótese de satisfazer as necessidades atuais, e também as futuras. Portanto, quem optou por desde logo carregar a informação existente, e não apenas a necessária, parabéns, tomou a decisão correta.

Poderiam aqui optar por juntar a data com o preço, ou então manter separada em duas estruturas:

```
typedef struct SData
{
        int ano, mes, dia;
} TData;


typedef struct SPreco
{
```

```
        double preco;
        TData inicio;
        struct SPreco *seguinte;
} TPreco;


typedef struct SProduto
{
        char *nome;
        TPreco *preco;
        struct SProduto *seguinte;
} TProduto;
```

Há duas entidades, o produto e o preço. Cada produto tem vários preços ao longo do tempo. Não há portanto que enganar, tem de existir duas estruturas, existindo um apontador de uma estrutura (produto) para a outra (preço). O preço tem também uma data.

Nesta alínea existe também a questão da ordenação. Como há uma só ordem, pode-se manter a lista na ordem desejada. Há também uma atividade formativa com a inserção de forma ordenada. Com listas há dois métodos bons que nada perdem devido a existir uma lista e não estar disponível o acesso arbitrário a qualquer posição. Isto é, pode haver acesso a qualquer posição, mas com custo em termos de tempo de acesso. Os dois métodos principais foram dados nas atividades formativas: merge sort e insert sort. Neste caso, se existe uma só ordenação, faz sentido utilizar o insert sort e manter a lista sempre ordenada, sendo também mais simples.

Bastaria adicionarem uma função (a utilizar no método de adição de um novo elemento):

```
// insere de forma ordenada elemento na lista
TProduto *PInsere(TProduto *lista, TProduto *elemento)
{
        if (lista == NULL || strcmp(lista->nome, elemento->nome) > 0) {
                // o primeiro elemento da lista
                elemento->seguinte = lista;
                return elemento;
        }
        lista->seguinte = PInsere(lista->seguinte, elemento);
        return lista;
}
```

Bem curto o código adicionado para permitir ter a lista ordenada quando comparado com algumas soluções.

O preço é apenas necessário ordenar na próxima alínea, mas pode ir já pelo mesmo caminho. No entanto convém fazer uma função de desigualdade entre datas, já que a expressão é longa.

```
int DataMenor(TData dataA, TData dataB)
{
        return dataA.ano < dataB.ano ||
                dataA.ano == dataB.ano && (
                        dataA.mes < dataB.mes ||
                        dataA.mes == dataB.mes &&
                        dataA.dia < dataB.dia);
```

```
}
TPreco *PInserirPreco(TPreco *preco, TPreco *novo)
{
        if (preco == NULL || DataMenor(preco->inicio, novo->inicio)) {
                // fica no início
                novo->seguinte = preco;
                return novo;
        }
        preco->seguinte = PInserirPreco(preco->seguinte, novo);
        return preco;
}
```

Como se pretende nesta alínea o número de alterações, bastaria uma função para calcular esse valor (e não uma variável para guardar esse registo):

```
int PAlteracoesPreco(TProduto*lista) {
        int resultado = 0;
        TPreco *preco = lista->preco;
        while (preco != NULL) {
                resultado++;
                preco = preco->seguinte;
        }
        return resultado;
}
```

Reparem na diferença. Quem utilizou uma variável, está a congelar a funcionalidade do programa. Quem leu tudo o que existe e constrói funções para calcular o que é preciso, vai permitir ao cliente mudar de ideias sem problema. Vamos supor que o cliente quer afinal apenas as alterações de preço desde 2010? Ou do último ano? Não tem problema, basta alterar esta função e adicionar um novo parâmetro, o ano a partir do qual as alterações de preço contam.

Já agora, houve também confusão em alguns trabalhos para ler o ficheiro e associar os preços ao último produto:

```
                if (isdigit(str[0]) && ultimoProduto!=NULL) { // novo preço
                        if (sscanf(str, "%d/%d/%d %lg",
                                &preco.inicio.ano,
                                &preco.inicio.mes,
                                &preco.inicio.dia,
                                &preco.preco) == 4)
                                PAdicionaPreco(ultimoProduto, preco.preco, p
reco.inicio);
                }
                else if (strlen(str)>0) // novo produto
                        lista = PAdiciona(lista, str, &ultimoProduto);
```

Convinha guardarem um apontador para o produto a adicionar. Ao verificarem que não têm algo, neste caso quando encontram uma linha de preço, não têm o produto, então criam uma variável para manter o valor que vão precisar depois. A falha em identificar a variável em falta (último produto), originou soluções um tanto ou quanto estranhas, como a estrutura de dados para conter todos os elementos.

Alinea C

Neste alínea, realizando a alínea B como deve de ser, não há nada de novo, apenas uma nova função para imprimir resultados, que percorre a estrutura de dados:

```
TPreco *preco;
printf("Produtos:");
while (lista != NULL) {
        printf("\n- %s;", lista->nome);
        preco = lista->preco;
        while (preco != NULL) {
                printf("\n  %d/%02d/%02d %.2f €",
                        preco->inicio.ano,
                        preco->inicio.mes,
                        preco->inicio.dia,
                        preco->preco);
                preco = preco->seguinte;
        }
        lista = lista->seguinte;
}
```

Alinea D

No ficheiro de conteúdo de exemplo, os produtos começaram a ser comercializados em 2010, existindo alterações de preços em 2012 e 2014. Em 2014 houve 5 alterações de preços, 4 num dos produtos e 1 alteração num outro produto. Mantenha a formação indicada no output de exemplo.

No relatório mensal deve indicar para o produto com o primeiro nome (assumindo ordem alfabética), informação mensal, e um índice de preços com informação mensal relativa a todos os produtos. Coloque um ano em cada linha e o mês em 12 colunas:

- para o primeiro produto, o preço médio por mês (caso o preço altere a meio do mês, fazer a média ponderada pelos dias, assumindo que um mês tem 30 dias. No caso do preço inicial ser nesse mês, considerar os dias iniciais do mês como estando no preço inicial);
- para o índice de preços, colocar a média de preços médios de todos os produtos em comercialização nesse mês (assumindo que os produtos valem o mesmo);

Conteúdo do ficheiro:

```
1
Produto B
2010/01/01 2.49
2012/01/01 2.59
```

```
Produto A
2012/01/01 1.74
2010/01/01 1.49
Produto C
2010/01/01 0.49
2014/01/01 0.99
2012/01/01 0.89
Produto D
2014/01/01 3.99
2014/04/01 4.09
2014/04/15 4.19
2014/04/25 4.29
2010/01/01 3.49
2012/01/01 3.89
```

Na primeira linha está um 1, pelo que é solicitado o segundo relatório.

Output esperado:

```
### Produto A ###
|  Ano |  Jan  |  Fev  |  Mar  |  Abr  |  Mai  |  Jun  |  Jul  |  Ago  |
Set  |  Out  |  Nov  |  Dez  |
| 2010 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |
1.49 |  1.49 |  1.49 |  1.49 |
| 2011 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |  1.49 |
1.49 |  1.49 |  1.49 |  1.49 |
| 2012 |  1.74 |  1.74 |  1.74 |  1.74 |  1.74 |  1.74 |  1.74 |  1.74 |
1.74 |  1.74 |  1.74 |  1.74 |
### Índice de Preços ###
|  Ano |  Jan  |  Fev  |  Mar  |  Abr  |  Mai  |  Jun  |  Jul  |  Ago  |
Set  |  Out  |  Nov  |  Dez  |
| 2010 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |
1.99 |  1.99 |  1.99 |  1.99 |
| 2011 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |  1.99 |
1.99 |  1.99 |  1.99 |  1.99 |
| 2012 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |
2.28 |  2.28 |  2.28 |  2.28 |
```

```
| 2013 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |  2.28 |
2.28 |  2.28 |  2.28 |  2.28 |
| 2014 |  2.33 |  2.33 |  2.33 |  2.37 |  2.40 |  2.40 |  2.40 |  2.40 |
2.40 |  2.40 |  2.40 |  2.40 |
```

Mostra-se o produto A já que é o primeiro produto, seguido dos anos em que o produto tem alterações de preços, o valor médio em cada mês. Nesta primeira parte, se existir um ou mais meses em que o produto não seja comercializado, colocar o preço médio como 0.00. Na segunda parte, mostrar a média de todos os produtos comercializados no mês. No caso de não existir qualquer produto a ser comercializado, preencher os espaços em falta com espaços (e não com o valor 0.00 como na parte 1).

Note que existe uma linha por cada ano. Se visualizar duas ou mais linhas por um só ano, é porque a largura do browser está a forçar a mudança de linha (pode baixar o zoom para ver as linhas sem estarem quebradas).

Note ainda que, de modo a obter valores exactamente iguais aos casos de teste, assuma para os preços valores do tipo double, e efectue a soma de todos os preços, e divida posteriormente pelo número de produtos, sem qualquer arredondamento, mostrando o resultado em 5 dígitos com precisão 2. No caso de registar diferenças, nos casos visíveis ou nos ocultos, se a diferença for explicada por algum problema com arredondamentos, não haverá lugar a penalizações, sendo os casos de teste falhados por este motivo contabilizados como bem sucedidos em termos de avaliação.

Os restantes casos de teste visíveis são também exemplos de execução.

Aqui era importante uma função auxiliar para obter os anos limite:

```
void PLimitesDatas(TProduto*lista, TData *minData, TData *maxData, int umProd
uto);
```

Para o relatório anual é importante uma função para dar o número de alterações:

```
int PAlteracoes(TProduto *lista, int ano);
```

A função do relatório anual, assenta com base nestas duas:

```
void MostraRelatorioAnual(TProduto*produto);
```

Para o outro relatório é conveniente uma função para retornar o valor médio por mês:

```
double PPrecoMedioMes(TProduto*produto, int ano, int mes);
```

Esta função teria de ter o cuidado de aceitar mudanças de preços dentro do mês. Tem que ter alguma atenção a nível de matemática e médias.

O relatório mensal pode ser feito com base nessas funções:

```
void MostraRelatorioMensal(TProduto*produto);
```

Esta era a função mais complexa, que poderia utilizar memória temporária para guardar as médias dos preços dos diversos produtos, para cada ano/mês que esteja em análise, bem como guardar o número de produtos nesse ano/mês. Teriam de ter o cuidado que quando não há produtos, teriam de colocar espaços e não 0 ou nan (resultado de 0.0/0).