Introdução

No âmbito da disciplina de EDA, foi-nos proposto a realização de um projeto prático, projeto este que consistia no desenvolvimento de uma máquina de vendas em C++. Para a realização eficiente deste projeto, foram criadas várias estruturas de forma a gravar os dados necessários em memória, e manter o código acessível. Este programa contém também várias funções com o objetivo de abordar os vários problemas que nos foram colocados, devidamente comentadas de maneira a facilitar a sua leitura. Este relatório tem como objetivo esclarecer as várias escolhas que foram tomadas ao longo das várias implementações encontradas ao programa.

Pré-Inicialização

Locale

Tendo em conta que o ficheiro de inicialização que contém os preços usa pontos em vez de vírgulas, e de forma a possibilitar o uso de caratéres especiais nas mensagens imprimidas na consola, a seguinte linha de código é usada.

```
locale::global(locale("English"));
```

Geração de números pseudo-aleatórios

Para tornar possível a geração de números pseudo-aleatórios é usada a função *srand*, incluída na biblioteca *stdlib.h*. A função *srand* tem como argumento um inteiro sem sinal, que servirá de semente. A semente escolhida será *time(NULL)*, que representa o número de segundos que se passaram após as 00:00 horas do dia 1 de Janeiro de 1970 UTC, quando esta função é executada.

Os números gerados pela função *srand* serão usados para escolher aleatóriamente na inicialização da máquina quantas moedas de cada tipo contém a caixa, e o número de slots, assim como a sua capacidade, preço e que produto contêm.

Adivinhar estas caraterísticas não apresenta nenhum problema de segurança à máquina de vendas, por isso não é necessário uma implementação mais complexa que permita gerar números mais díficeis de adivinhar.

```
srand(time(NULL));
```

Referências:

http://www.cplusplus.com/reference/cstdlib/srand/

http://www.cplusplus.com/reference/ctime/time/

Ficheiros de inicialização

Para a máquina de vendas ser inicializada corretamente, foi imposto que esta retirasse os produtos e os preços de dois ficheiros correspondentes. Visto que a máquina de vendas tem que ser inicializada sempre que o programa se inicia, o programa pede a localização de ambos os ficheiros quando começa, ou as localizações são passadas como argumento.

```
Localizaçao do ficheiro que contém os produtos: C:\Users\Mistakx\Desktop

Localizaçao do ficheiro que contém os preços:C:\Users\Mistakx\Desktop
```

Abertura dos ficheiros de inicialização

Após o utilizador inserir a localização de ambos os ficheiros, estes ficheiros têm que ser abertos pelo programa, e tanto os produtos quanto os preços de lá retirados. Existem várias formas de abordar o problema, nomeadamente:

- Abir os ficheiros sempre que necessário.

  Vantagens:
  - Não usa memória RAM para armazenar os ficheiros, pois são ambos mantidos no disco.

  Desvantagens:
  - É mais lento pois o programa precisa de abrir o arquivo mantido no disco sempre que necessita de aceder a informações. Até a velocidade de leitura dos SSDs não é tão rápida quanto a velocidade da memória RAM.
  - Não permite que os ficheiros de inicialização estejam guardados num dispositivo externo, pois se o despositivo for removido, não é possível aceder aos ficheiros.

- Abrir os ficheiros uma vez, e guardar os seus conteúdos na memória RAM.

  Vantagens:
  - É mais rápido.
  - Permite que os ficheiros de inicialização estejam gravados num dispositivo externo.

  Desvantagens:
  - Usa memória RAM para armazenar os ficheiros.

A abordagem escolhida foi a segunda, visto que cada produto e cada preço ocupam uma quantidade negligível na memória de acesso aleatório, sendo necessários documentos de inicialização muito grandes e pouco realistas para começarem a ser considerados um problema.

Alocação de memória para gravação dos ficheiros

Para guardar os documentos na memória RAM foram criadas:

- Duas funções, com o objetivo de fazer "*parsing*" aos documentos de inicialização.

```
void prices_text_parsing(string file_path, Prices* prices){ ... }

void products_text_parsing(string file_path , Products* products){ ... }
```

- Duas estruturas, onde o resultado de ambos os "*parsings*" serão gravados.

```
struct Products {
    string* array = 0; // The array that contains each product.
    int lenght = 0; // The array lenght.
};

struct Prices {
    float* array = 0; // The array that contains each price.
    int lenght = 0; // The array lenght.
};
```

Visto que cada produto e cada preço serão gravados num array correspondente, temos que alocar memória para este array. Temos duas hipóteses:

- Percorrer ambos os documentos de inicialização. Gravar o número de elementos em cada documento numa variável. Inicializar os dois arrays com o número encontrado de elementos.

```
prices->array = new float[prices->lenght];

products->array = new string[products->lenght];
```

Vantagens:

- Garante que os arrays alocam apenas a memória necessária para gravar os elementos dos documentos de inicialização.

Desvantagens:

- Tem que percorrer cada documento duas vezes, uma para descobrir o número de elementos, e outra para o gravar no array.

- Alocar um grande espaço de memória que nos garante com quase 100% de certezas que os ficheiros de inicialização cabem lá dentro.

```
products->array = new string[10000];

prices->array = new float[10000];
```

Vantagens:

- Apenas percorre cada documento uma vez, para colocar os seus elementos no array correspondente.

Desvantagens:

- Se a memória a alocar escolhida for pouca, corre-se o risco de não ser suficiente para gravar todos os elementos pretendidos. Se for muita, desperdiça-se memória.

Observações:

- Se o número de produtos/preços contidos nos documentos de inicialização for conhecido, esta implementação é mais rápida que a primeira, com exatamente os mesmos custos no que toca à memória RAM.

Neste caso, o uso de listas ligadas seria a melhor implementação.

Vantagens:

- Apenas percorre cada documento uma vez, para colocar cada elemento num node correspondente.

- Não seria necessária a implementação das estruturas anteriormente referidas. Enquanto que ao percorrer um array é necessário saber o seu tamanho (para saber quando parar de o percorrer), as listas ligadas não têm este requisito, pois o último elemento aponta sempre para NULL.

Desvantagens:

- Nenhuma.

A implementação escolhida foi a primeira. Tendo em conta que os arrays permanecerão em memória ao longo da execução do programa, e que o carregamento dos ficheiros apenas ocorre no início da mesma, é preferível que a máquina demore um pouco mais tempo a ligar mas garanta um uso eficiente da sua memória RAM (memória esta muitas vezes bastante pequena em sistemas embarcados como os usados em máquinas de vendas).

Detalhes sobre a implementação escolhida

A escolha da primeira implementação trás consigo alguns detalhes:

Todas as streams contém informação sobre o seu estado atual. Quando a função getline acaba de percorrer cada documento de inicialização pela primeira vez, ela adiciona uma "*flag*", com o nome de *eofbit* (abreviatura para End-of-File Bit). Se esta flag não for limpa, para voltar a ler o ficheiro, seria necessário fechá-lo e abri-lo outra vez. Para evitar este processo desnecessário, é limpa a flag, e a posição do próximo caractér a ser limpo é redifinida, de forma a ler o primeiro caractér do ficheiro outra vez.

```
file.clear(); /* This clears the eof flag. While checking the documentation we found that C++ behaviour regarding this line and the next has changed.
    C++98 - If the eofbit flag is set before the seekg call, the function fails.
    C++11 - The seekg function clears the eofbit flag, if set before the call.
    !Reference: http://www.cplusplus.com/reference/istream/istream/seekg/
    Conclusion: This line isn't needed in the newer versions of C++.
*/
file.seekg(0, ios::beg); // Sets the position of the next character to be extracted from the input stream back at the beginning so we can getline again.
```

A partir do C++11 a função *seekg* já limpa a flag se esta for encontrada, eliminando a necessidade de escrever a linha anterior.

Referências:

http://www.cplusplus.com/reference/istream/istream/seekg/

https://stackoverflow.com/questions/5343173/returning-to-beginning-of-file-after-getline

O ficheiro dos preços contém números com casas décimais, em contraste ao ficheiro dos produtos que contém palavras. Tendo isto em conta cada linha do ficheiro dos preços tem que ser convertida para *float* antes de ser adicionada ao array que contém os preços.

```
prices->array[i] = stof(line); // Converts line to float before adding to array.
```

Referências:

http://www.cplusplus.com/reference/string/stof/

Erros ao abrir/"parsing" dos ficheiros

Se por alguma razão algum ou ambos os ficheiros de inicialização não forem abertos corretamente, ou o ficheiro dos preços não tiver nenhum preço, o seguinte erro é mostrado, e a máquina não é inicializada.

```
Ocorreu um erro ao tentar abrir "C:\Users\Mistakx\Desktop\produto.txt".
O ficheiro nao existe ou está a ser usado por outro processo.

Ocorreu um erro ao tentar abrir "C:\Users\Mistakx\Desktop\preco.txt".
O ficheiro nao existe ou está a ser usado por outro processo.

Os ficheiros de inicializaçao da máquina nao foram abertos corretamente.
```

Inicialização da máquina de vendas

Se ambos os ficheiros forem abertos corretamente e o ficheiro dos preços tiver algum preço a máquina é inicializada.

Alocação de memória para a máquina de vendas

A máquina de vendas é uma estrutura que contém um array de slots (outra estrutura), o número de slots, e a caixa. A caixa é um array que contém o número de cada moeda nela presente, ordenadas por ordem crescente. 5 cêntimos, 10 cêntimos, 20 cêntimos, 50 cêntimos, 1 euro, 2 euros.

```
struct Vending_machine {
    Slot* slots = 0; // The machine itself is an array of slots.
    int size = 0; // The number of slots the machine contains.
    int cash_box[6]{}; // The number of coins the machine contains. This array is sorted by ascending order: 5 cents, 10 cents, 20 cents, 50 cents, 1 euro, 2 euros.
};
```

Um slot é caraterizado pela sua letra, produto que contém, quantidade de produtos nele presentes, preço e pela sua capacidade. Isto é refletido na sua estrutura.

```
struct Slot {

    char letter = 0; // The letter of the slot
    string product = "Vazio"; // The product that is currently in the slot.
    float price = 0; // The current price of the slot.
    int quantity = 0; // The number of products currently in the slot.
    int capacity = 0; // The maximum capacity of the slot.
};
```

Inicialização dos slots

É gerado um número pseudo-aleatório entre 9 a 12 slots (incluindo ambos) que será igual ao número de slots a serem inicializados.

```
// Initializes the vending_machine size.
vending_machine->size = rand() % 4 + 9; // Generates a random number between 9 and 12 (including both)
```

Logo de seguida é criado um array de slots dinâmicamente alocado com o tamanho gerado, e são inicializados todos os slots.

```
// Initializes each vending machine slot.
vending_machine->slots = new Slot[vending_machine->size];

for (int i = 0; i < vending_machine->size; i++) {

    initialize_slot(&vending_machine->slots[i], i, products, prices);

}
```

As caraterísticas dos slots são todas inicializadas pela função *initialize_slot*.

```
void initialize_slot(Slot* slot, int slot_position, Products* initialization_products, Prices text_prices) {

    slot->letter = char(slot_position + 65); // Initializes the slot with it's corresponding letter.
    slot->product = choose_random_product(initialization_products); // Choose a random product to place on the slot from the remaining products.
    slot->price = text_prices.array[rand() % text_prices.lenght]; // The slot starts with a random price chosen from the prices.txt file.
    slot->capacity = rand() % 6 + 5; // Initializes the slot with a random capacity (between 5 and 10 (including both))
    slot->quantity = slot->capacity; // The slot starts full of products.
```

Letra

Em C++, é possível obter o valor décimal correspondente na tabela ASCII de cada caractér, e vice-versa. Sabendo isto, usa-se *int(char('A'))* para saber o valor décimal do caractér 'A', que é 65. Sabemos agora que ao somar 65 a uma dada posição da máquina, e convertendo o resultado para char, temos a letra correspondente à posição escolhida.

Produto

Um dos requisitos era a máquina não ser inicializada com produtos repetidos. Para tal, foi criada a função *choose_random_product* que retorna aleatóriamente um dos produtos do array, removendo-o do array logo de seguida para que este não se repita futuramente. Esta remoção recorre à função *delete_product*.

A função *delete_product*cria um novo array, igual ao anterior, mas com a posição escolhida removida, e substitui o antigo array por este novo, eliminando o antigo para não causar nenhum "memory leak". O uso de listas ligadas permitiria simplesmente eliminar um dos seus nós, sendo este um processo muito mais fácil e eficiente do que o implementado.

No caso do ficheiro dos produtos não ter um número de produtos suficientes para permitir uma inicailização correta, a função *choose_random_product* mostra o seguinte erro.

```
Ocorreu um erro ao escolher um produto para introduzir na máquina.
Verifique se o ficheiro que contém os produtos tem um número de produtos suficientes.
```

O preço é aleatóriamente escolhido entre os preços gravados no array criado na abertura dos ficheiros, podendo se repetir.

Capacidade

A capacidade de cada slot é aleatóriamente gerada, entre 5 e 10.

Quantidade

A quantidade de produtos quando a máquina é inicializada é igual à capacidade, pois todos os slots começam cheios de produtos.

Menu Principal

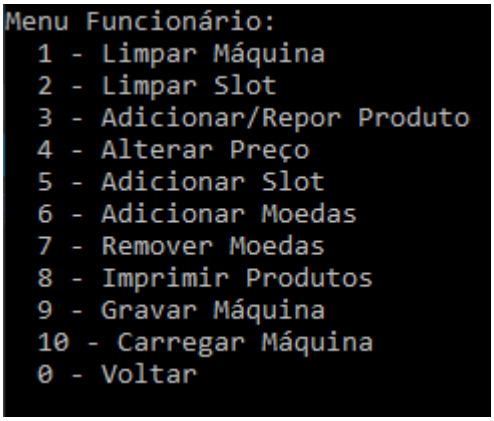
Se a inicialização da máquina decorre sem nenhum problema as estruturas que guardam os ficheiros de inicialização são eliminadas, e o programa abre o seu menu principal, onde o utilizador dirá à máquina de vendas se é um funcionário ou um cliente.

```
Modo de utilizaçao:
  1 - Funcionário
  2 - Utilizador
```

O menu principal é um ciclo que limpa a consola sempre que começa. Se esta limpeza não for efetuada, ao regressar do menu do funcionário ou do menu do utilizador, as opções destes menus continuariam na consola, o que não é desejável. A limpeza é realizada recorrendo ao comando *cls*, comando este pertencente ao *cmd.exe* do Windows. O seu equivalente noutras consolas/terminais é mais frequentemente o comando *clear*.

Funcionário

Se o utilizador escolher a opção “Funcionário” no menu principal é levado para o seguinte menu.



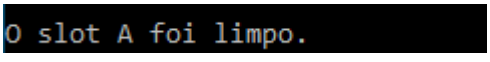
Limpar Máquina

Limpa todos os slots contidos na máquina, usando a função *clean_slot* em cada um.

Limpar Slot

Abre um menu que pede ao funcionário a letra do slot a limpar.

Se o slot for limpo com sucesso, mostra a seguinte mensagem.



Se o slot não existir, mostra o seguinte erro.

```
Nao existe nenhum slot na posiçao K.
```

Se o slot já estiver limpo, mostra o seguinte erro.

```
O slot A já se encontra limpo.
```

A limpeza do slot é feita pela função *clear_slot*. Esta função altera a quantidade do slot para 0, o preço para 0, e o produto que nele se encontra para "Vazio". A capacidade do slot é mantida.

```
void clean_slot(Slot* slot) {

    slot->quantity = 0;
    slot->product = "Vazio";
    slot->price = 0;

}
```

Adicionar/Repor Produto

Abre um menu que pergunta ao funcionário o slot em que pretende adicionar/repor produto.

```
Slot a repor:
```

Escolha de um slot vazio

Se o funcionário escolher um slot vazio, abre um menu que pergunta o nome do produto que pretende adicionar a este slot vazio.

```
Nome do produto:
```

Se o nome do produto inserido for "Vazio" é mostrada a seguinte mensagem.

```
O nome do produto nao pode ser "Vazio".
```

Se o nome do produto inserido já existir, então o slot terá o seu preço alterado para o preço do produto já existente.

Após o funcionário escolher um nome válido para o produto, é lhe pedido para indicar o número de produtos que pretende inserir ao slot vazio. Apesar dos slots estarem ordenados, o algoritmo de procura escolhido foi o de procura sequencial por razões que serão abordadas posteriormente.

```
Número de produtos a adicionar:
```

Se o número introduzido não for válido é mostrada a seguinte mensagem.

```
Número de produtos a adicionar:
```

Escolha de um slot vazio: Se o produto inserido já existe noutro slot

Se o número de produtos inseridos couber no slot vazio os produtos serão lá guardados, e o preço do slot será alterado de forma a corresponder com o preço do produto já existente, para evitar ter dois produtos com o mesmo preço.

Se o número de produtos não couber no slot escolhido, é procurado um slot vazio que os suporte, e o preço do slot será alterado de forma a corresponder com o preço do produto já existente, para evitar ter dois produtos com o mesmo preço.

Escolha de um slot vazio: Se o produto inserido não existe noutro slot

Se o número de produtos inseridos couber no slot vazio os produtos serão inseridos neste slot vazio, e o preço do slot não será alterado.

Se o número de produtos não couber no slot escolhido, é procurado um slot vazio que os suporte, e o preço do slot não será alterado.

Escolha de um slot não vazio

Se o funcionário escolher um slot que já contém produtos o nome do produto a repor não lhe é perguntado. É lhe apenas pedido a quantidade de produtos que deseja repor.

```
Número de produtos a repor:
```

Se o número introduzido não for válido é mostrada a seguinte mensagem.

```
Introduza um número válido de produtos a repor.
```

Se o número de produtos inseridos couber no slot vazio os produtos serão inseridos neste slot vazio.

Se o número de produtos não couber no slot escolhido, é procurado um slot vazio que os suporte.

Procura de um slot vazio que suporte os produtos.

Se o número de produtos a adicionar/repor exceder a capacidade do slot escolhida é apresentada a seguinte mensagem.

```
A quantidade de produtos que tentou introduzir nao cabe no slot escolhido.
```

De seguida é procurado um slot vazio que suporte o número de produtos inseridos. Se for encontrado um slot vazio, usando a função *add_to_fitting_slot,* os produtos lá serão colocados. Se este não for o caso, é mostrada a seguinte mensagem.

```
Nao foi encontrado nenhum slot vazio com capacidade suficiente para suportar o número de produtos inseridos.
```

Opções do funcionário

Se após este processo inteiro os produtos não couberem nem no slot escolhido, nem ter sido encontrado um slot vazio que os suportasse, um dos seguintes menus é mostrado ao funcionário.

Se o slot escolhido já se encontrar cheio, usando a função *employee_choices_full.*

```
O que deseja fazer:
  1 - Alterar a capacidade do slot A de forma a acomodar todos os produtos inseridos.
  0 - Voltar
```

Se o slot escolhido não se encontrar cheio, usando a função *employee_choices_not_full.*

```
O que deseja fazer:
  1 - Alterar a capacidade do slot A de forma a acomodar todos os produtos inseridos.
  2 - Inserir apenas os produtos que conseguem ser inseridos tendo em conta a capacidade do slot A.
  0 - Voltar
```

Alterar preço

Abre um menu onde o funcionário escolhe o nome do produto que cujo preço deseja alterar. Antes do programa proceder e pedir o novo preço, é verificado se este produto existe na máquina. Este é um processo mais demorado, pois tem que percorrer a máquina duas vezes (uma para verificar se o produto existe e outra para alterar o preço) mas evita que o utilizador introduza o preço em vão se o produto não existir na máquina.

Uma alternativa seria procurar se o produto existe, e, se existir, guardar a localização dos slots onde este se encontra para mais tarde serem mudados os preços. Esta implementação foi considerada, mas os ganhos seriam neglígiveis neste exato programa devido a vários fatores (baixo número de slots, depêndencia no utilizador por parte do programa, tempo de impressão da consola/terminal).

```
Nome do produto cujo preço deseja alterar:
```

Se o produto inserido não existe na máquina é imprimido o seguinte erro.

```
O produto inserido nao se encontra na máquina.
```

Se o nome inserido for “Vazio” é imprimido o seguinte erro. Algo que pode ser implementado futuramente, se for justificável, é não deixar que o nome seja nenhuma variação da palavra “Vazio”, como por exemplo “VAZIO” ou “vazio”. Esta implementação transformaria a string introduzida pelo utilizador numa string igual com todas as letras maiúsculas, e verificaria se o resultado é a *string* “VAZIO”. Isto aumentaria bastante a complexidade do código, pois o C++ não inclui nenhuma maneira fácil de maiusculizar *strings*, apenas caractéres.

```
O nome do produto nao pode ser "Vazio".
```

Se o produto inserido existir na máquina, é pedido o preço que deseja atribuír ao produto.

```
Preço que deseja atribuír a Lays:
```

Se atribuír um preço negativo é imprimido o seguinte erro.

```
Por favor introduza um preço positivo.
```

Se atribuír um preço que não é multiplo de 5 cêntimos é imprimido o seguinte erro.

```
Por favor introduza um valor múltiplo de 5 cêntimos.
```

Se atribuír um preço válido, todos os slots que contém produtos com o nome escolhido terão o seu preço mudado para o preço inserido.

Adicionar Slots

Abre um menu onde o funcionário escolhe a letra correspondente ao slot que pretende adicionar. Se forem introduzidas letras minúsculas elas serão maiusculizadas.

```
Letra correspondente ao slot que pretende adicionar:
```

Se o funcionário não introduziu uma letra entre A e Z é imprimido o seguinte erro.

```
Por favor introduza uma letra entre A e Z
```

Se o slot que introduziu já existe é imprimido o seguinte erro. A procura é feita usando o algoritmo de procura sequencial. Visto que os slots estão ordenados poderia ser usada uma procura binária, no entanto esta não foi a implementação escolhida . É de notar que C++ é uma línguagem extremamente rápida. Com no máximo 26 slots para procurar, o programa passará muito mais tempo à espera do utilizador e dependente do tempo de resposta da consola/terminal do que a executar qualquer algoritmo de procura/ordenação.

```
O slot que introduziu já existe.
```

Se foi introduzida uma letra entre A e Z, e o não existe um slot correspondente à letra introduzida, é pedida a capacidade do novo slot.

```
Capacidade do novo slot (Slot M):
```

Após ser inserida a capacidade do slot a ser inserido, é criado um novo array de slots, igual ao anterior, mas com mais um slot no fim, com a letra escolhida. O antigo array é substituído por este novo array, eliminando o antigo para não causar nenhum *memory leak*. O uso de listas ligadas permitiria simplesmente eliminar um dos seus nós, sendo este um processo muito mais fácil e eficiente do que o implementado.

Da primeira vez que um slot é inserido, é nos garantido que os slots continuam em ordem. A partir do segundo slot inserido, passamos a ter que os ordenar. Poderia ser usada uma implementação que tem isto em conta mas visto que ordernar os slots é um processo que não afeta o desempenho do programa, pelas razões que vão ser referidas abaixo, sempre que é adicionado um slot a máquina é ordenada.

O algoritmo de ordenação usado ao longo do programa é o insertion sort. Com no máximo 26 produtos para ordenar, a consola/terminal demorará muito mais tempo a imprimir os produtos ordenados, do que o algoritmo de sort demorará a ordená-los. Não se justifica a implementação de um algoritmo mais complexo. O insertion sort tem também a vantagem de ser um algoritmo com complexidade de memória igual a um, um fator importante pelas razões realçadas anteriormente.

Adicionar Moedas

Abre um menu onde o funcionário escolhe a quantidade de cada moeda que pretende adicionar. Estas moedas são imediatamente introduzidas na caixa. Após as moedas serem adicionadas, a função *check_funds* verifica se a caixa têm 3 ou menos de cada moeda, e imprime uma mensagem se este for o caso.

Remover Moedas

Abre um menu onde o funcionário escolhe a quantidade de cada moeda que pretende remover. Se houverem moedas suficientes para serem removidas, estas são imediatamente removidas da caixa. Se não houverem moedas suficientes, é imprimido um erro.

```
A máquina nao tem 100 moedas de 2 euros.
```

Após as moedas serem ou não removidas, a função *check_funds* verifica se a caixa têm 3 ou menos de cada moeda, e imprime uma mensagem se este for o caso.

Imprimir produtos

Se não houver nenhum produto na máquina é imprimido o seguinte erro. É usada a função *vending_machine_is_empty* para fazer esta verificação.

```
Nao é possível imprimir nenhum produto pois a máquina está vazia.
```

Se houverem produtos é imprimido o seguinte menu.

```
Imprimir produtos:
  1 - Por ordem alfabética.
  2 - Por preço.
  3 - Por quantidade disponível.
```

Ordem alfabética

Para ordernar por ordem alfabética, é criado um array de strings. De seguida são copiados para este novo array os produtos em cada slot da máquina que ainda nele não se encontram, de forma a não imprimir duas ou mais vezes o

mesmo produto. Para tal é usada a função *is_in_array*. Após a cópia, o array é ordenado. Os operadores de comparação em C++ podem ser usados para verificar se uma string vem antes de outra no dicionário, por exemplo “Abel” < “Teresa” é uma expressão que retorna 1.

O tamanho do array criado será o tamanho da máquina de vendas. No caso da máquina de vendas ter produtos iguais em slots diferentes, este array terá espaços desnecessários. Uma implementação possível seria descobrir o número de produtos antes de ser alocado o array. Sabendo que não é algo comum haver produtos repetidos, que no máximo o array terá 26 elementos, e que este array será eliminado imediatamente após os produtos serem imprimidos, esta implementação é desnecessária. Esta possível implementação seria também mais lenta.

Para imprimir o array criado é usada a função *print_array.*

Preço

Para ordenar por preço, é criado um array de slots. Ao contrário da implementação anterior, não é possível criar apenas um array de floats, pois é necessário ordenar ambos os preços e os nomes dos produtos. Poderia ter sido uma nova estrutura que contivesse apenas o preço e o nome do produto, mas isto é desnecessário pois a estrutura *Slot* já contém ambos.

De seguida são copiados para este novo array os slots que contêm produtos que ainda nele não se encontram, de forma a não imprimir duas ou mais vezes o mesmo produto. Para tal é usada a função *is_in_array*. Após a cópia, o array é ordenado.

A implementação em relação ao tamanho do array criado é a mesma que a anterior.

Quantidade

Para ordenar por quantidade, é criado um array de slots, pelas mesmas razões da ordenação por preço. Se houverem produtos duplicados, em vez dos slots que os contêm serem todos copiados para o novo array, apenas é copiado o primeiro, e à sua quantidade vai sendo somada a quantidade de produtos repetidos. Após a cópia, o array é ordenado.

A implementação em relação ao tamanho do array criado é a mesma que a anterior.

Gravar Máquina

Abre um menu onde o funcionário escolhe a localização onde pretende gravar a máquina.

```
Gravar a máquina na localizaçao:
```

Se ocorrer um erro ao gravar a máquina, é imprimida a seguinte mensagem.

```
cout << "Ocorreu um problema ao tentar abrir \"" << file_path << "\"."

<< endl << "A localização é inválida ou o ficheiro está a ser usado por outro processo." << endl;
```

Se a máquina for gravada com sucesso imprime a seguinte mensagem.

```
A máquina foi gravada em "C:\Users\Mistakx\Desktop\Testing123.txt".
```

A máquina é gravada no seguinte formato. Poderiam apenas ter sido gravados os valores, mas indicar o que é que cada número corresponde permite ao funcionário fácilmente alterar as propriedades da máquina gravada, quer seja mudar um preço, ou até adicionar um slot. A diferença entre o espaço do documento produzido por esta implementação, e a implementação escolhida é muito pequeno. É de notar que o documento não será guardado na memória RAM, logo em príncipio o seu tamanho não será problema.

```
Vending Machine Size: 11
Cash Box - 2 euros: 12
Cash Box - 1 euro: 10
Cash Box - 50 cents: 11
Cash Box - 20 cents: 14
Cash Box - 10 cents: 19
Cash Box - 5 cents: 14

Slot Letter: A
Slot Capacity: 8
Current Number of Products: 8
Product Name: Compal Manga Laranja
Slot Price: 1.2
```

Após o documento ser guardado, é lhe concedido o atributo "*read-only*", de forma a evitar que um funcionário desatento cause alterações não desejadas ao ficheiro. Para tal é usado o comando do Windows a*ttrib +r*. Em sistemas UNIX o comando seria *chmod 444*.

```
system(("attrib +r " + file_path).c_str()); // Changes the file to read only. Only works on windows.
```

Referências:

https://stackoverflow.com/questions/7416445/what-is-use-of-c-str-function-in-c

https://stackoverflow.com/questions/4907805/using-variables-in-system-function-c

Carregar Máquina

Abre um menu onde o funcionário escolhe a localização da máquina que pretende carregar.

```
Carregar a máquina gravada na localizaçao:
```

Se ocorrer um erro ao abrir o ficheiro que contém a máquina imprime a seguinte mensagem.

```
Ocorreu um erro ao tentar abrir "C:\Users\Mistakx\Desktop\Test".
O ficheiro nao existe ou está a ser usado por outro processo.
```

Se o ficheiro for aberto a máquina é carregada. É usado o método *substr* para extraír os valores de cada linha. Alguns valores são *floats*, sendo necessária a função *stof* anteriormente referida, outros são inteiros, sendo necessária a função *stoi.* Após os valores serem extraídos do ficheiro, é imprimida a seguinte mensagem.

```
A máquina "C:\Users\Mistakx\Desktop\Testing123.txt" foi carregada.
```

Referências:

http://www.cplusplus.com/reference/string/string/substr/

http://www.cplusplus.com/reference/string/stoi/

Voltar

Volta ao menu principal.

Cliente

Se o utilizador escolher a opção “Cliente” no menu principal é levado para o seguinte menu.

```
O que deseja fazer:
  1 - Comprar Produto
  0 - Voltar
```

Comprar Produto

Abre um menu onde o funcionário escolhe o slot que contém o produto que pretende comprar. Se forem introduzidas letras minúsculas elas serão maiusculizadas.

```
Introduza o slot:
```

Se o slot introduzido não existe mostra a seguinte mensagem.

```
O slot que escolheu nao existe.
```

Se o slot introduzido existir e estiver vazio mostra a seguinte mensagem.

```
O slot escolhido está vazio.
```

Se o slot introduzido existir e não estiver vazio, pede para introduzir as moedas.

```
Escolheu: Compal Laranja
Preço: 2

Introduza a quantidade de moedas de 2 euros:
```

Se introduziu dinheiro suficiente, e a máquina tem troco suficiente, então devolve o produto. O troco é também devolvido e imprimido no ecrã usando a função *give_change*. Esta função tenta dar ao cliente o menor número de moedas possível.

```
 ------------------------- Troco -------------------------
| 2 euros   1 euro   50 cent   20 cent   10 cent   5 cent |
| 0         1        0         1         0         0      |
 ---------------------------------------------------------

Valor Introduzido: 2

Produto devolvido.
```

Se introduziu dinheiro suficiente, mas a máquina não tem troco suficiente, então não devolve o produto, devolve o dinheiro inserido e mostra o seguinte erro.

```
Produto nao devolvido. Nao existe troco suficiente na máquina.
```

Se não foi introduzido dinheiro suficiente então não devolve o produto, devolve o dinheiro inserido e mostra o seguinte erro. Poderia ser implementado um algoritmo que dissesse o valor que falta. Por exemplo: Faltaram 2 euros. Faltaram 1.20 euros. Faltaram 0.50 cêntimos. Faltou 1 euro. Faltou 1 cêntimo. Todos estes casos teriam de ser considerados. Visto que o valor introduzido já é exposto no ecrã, é debatível se tal é ou não necessário.

```
Produto nao devolvido. Nao introduziu dinheiro suficiente.
```

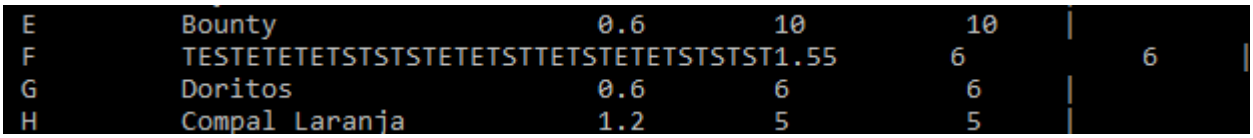
Voltar

Volta ao menu principal.

Imprimir máquina de vendas

A cada escolha que o utilizador faz, a consola tem que ser limpa de maneira a garantir uma fácil utilização da máquina de vendas. Visto que queremos que a máquina de vendas permaneça vísivel, e que vá sendo atualizada ao longo da execução do programa, foi criada a função *refresh_console* com este exato objetivo. É debatível se os utilizadores deviam poder ver a caixa da máquina de vendas, mas acreditamos ser uma mais valia. Não só torna mais fácil perceber como a máquina está a funcionar (importante neste projeto), torna possível usar a máquina para destrocar moedas, algo que a mim dava jeito numa máquina real.

A máquina de vendas é imprimida de forma a que cada slot fique numa linha, facilitando imenso a leitura do utilizador. Para tal, recorre à função *setw.* Se algum produto tiver um nome grande o suficiente, a sua linha será desconfigurada.



De maneira a reduzir a probabilidade disto acontecer foi dado um espaço grande entre a coluna dos produtos e a coluna dos preços. Uma implementação possível seria encontrar o tamanho do produto com o maior nome, e ajustar a distância das colunas em função do tamanho encontrado, sendo este um processo mais lento.

Referências:

http://www.cplusplus.com/reference/iomanip/setw/

Possíveis melhorias

Apesar de terem sido realizadas várias validações ao input do utilizador ao longo do programa, este processo poderia ser extendido.

- Os ficheiro dos preços poderia sofrer uma validação, de forma a não conter letras.
- Ao introduzir um nome do produto, considerar variações na maiusculização deste nome como o mesmo produto. “KIT-KAT” = “kit-kat”. Isto implicaria criar uma função que transformasse uma string numa string igual com todas as letras maiúsculas, tendo em conta que a função *toUpper* já incluída no C++ recebe um inteiro como argumento.
- Cortar espaços que possam ser introduzidos por engano no início/fim do nome do produto. “Kit-Kat “ != “Kit-Kat”.
- Guardar em memória os resultados da ordenação dos produtos, de forma a não ter que voltar a ordená-los se o funcionário os quiser voltar a ver, e estes ainda não tiverem sido alterados. Visto que a ordenação é rápida, e o espaço em memória RAM é pouco, esta implementação é contra-produtiva. A ordenação poderia também ser gravada em disco, mas provavelmente seria mais rápido ordenar outra vez do que estar a abrir um ficheiro.

Conclusão

Com base no conhecimento obtido tanto nas aulas teóricas quanto nas práticas, foi possível realizar este primeiro projeto com extremo sucesso. O desenvolvimento do projeto obrigou a serem aprofundandos conhecimentos àcerca de estruturas de dados e algoritmos, entre estes a alocação de memória dinâmica, e as diferenças entre os vários algoritmos de ordenação. Assim sendo, acreditamos que este projeto nos tornou mais proficientes não só em C++, mas em tudo o que toca à programação em geral.