

Estruturas de Dados e Algoritmos Projeto prático 2 - 2019/2020

 ***Trabalho realizado por:***

Pedro Rodrigues - 2046619

Diogo Santos - 2042919

João Cupido - 2046519

Michael Costa - 2048919

Paulo Drumond - 2048719

***1º Ano de Engenharia Informática***

Índice

[**Introdução**](#_xlqlthicd68x) **3**

[**Implementação**](#_zgjr75d889n) **4**

[main.cpp](#_rgubba2kwof0) 4

[Fila.h](#_bvzl2ruwtdy) 5

[Fila.cpp](#_d45tsmo71cym) 5

[inicio.h](#_kb3epk59zmgl) 7

[inicio.cpp](#_p4ofdzobm572) 7

8

8

9

9

[**Conclusão**](#_3nhxd35v6mfw) **11**

# 

# Introdução

Neste projeto, criámos um programa em C++ através do Visual Studio com o objetivo final de simular o funcionamento da empresa “​ExpressoEDA​”. Esta é uma empresa de transporte de passageiros de ponto a ponto na qual não podem entrar passageiros nas paragens intermédias, apenas sair.

O projeto deverá simular todas as funcionalidades relativas ao funcionamento desta empresa, de uma forma geral a entrada de passageiros na origem e a respetiva saída de passageiros nos seus destinos.

Tivemos contudo certas restrições relativas á construção do projeto. Foi-nos proibido o uso da classe/biblioteca *vector*, e pedido o uso dos tipos de dados definidos pelos alunos, structs, listas ligadas e árvores de pesquisa binária.

É também importante mencionar que cada viagem é levada a cabo por um autocarro com uma capacidade de passageiros variável entre 5 e 10 inclusive, um motorista, e uma matrícula. O motorista é identificado pelo seu primeiro e último nome. A matrícula é composta por 4 valores hexadecimais aleatórios. Cada autocarro encontra-se associado a uma lista de passageiros, todos os autocarros iniciam a sua viagem com todas as vagas preenchidas passando por um número variável de paragens entre 4 e 9 inclusive.

// ainda por rever(apenas a introdução está feita)

# 

# Implementação

## main.cpp

O ficheiro main.cpp é constituído por uma função apenas, *main*, que tem como objetivo ligar os restantes itens numa função apenas e controlá-los de acordo com as indicações e desejos do utilizador.

No início, fora da função *main*, e juntamente com os *#include* e outros, é definida uma variável global *int numSlots*. Esta variável é referente ao número de slots da máquina, e tem o propósito de fazer com que este esteja sempre atualizado em qualquer situação do decorrer do programa, também possibilitando uma atualização mais fácil da mesma, ao longo de funções distintas, que poderiam ficar com valores diferentes para a sua variável local numSlots, em vez de corretamente atualizá-la para uma nova máquina que seja editada ou carregada no decorrer do programa.

Seguidamente, na inicialização, definimos o locale como português, para podermos imprimir símbolos especiais para a consola, juntamente com a definição de uma semente que será usada para gerar valores aleatórios através do *srand((unsigned)time(NULL))*. Verificamos o caso de haver argumentos passados pela linha de comandos, e, se existirem, inicializamos a máquina de acordo com esses. Noutro caso, o programa cria uma nova máquina com um número de slots aleatório no intervalo estipulado, e uma nova caixa de moedas, com quantidades também aleatórias para cada tipo de moeda definida.

A máquina em si é definida como um *array de slots*, estruturas que são definidas pelo seu código, quantidade, capacidade (*quantidadeMax*), e o seu produto, que por sua vez é definido como tendo um nome e um preço. Esta máquina é inicializada com o tamanho de 100, para que haja suficiente memória alocada para possibilitar uma adição e remoção de *slots* mais simples no futuro.

Se for feita passagem de argumentos pela linha de comandos, para efeitos de um carregamento correto da máquina, o primeiro argumento deverá ser o caminho do ficheiro referente ao conteúdo dos slots, o segundo à caixa de moedas, e o terceiro referente ao número de slots na máquina.

Após criada ou carregada a máquina, é apresentado ao utilizador um menu com a escolha entre cliente ou funcionário, que irá levar a que sejam executadas as funções referentes a esses dois tipos de utilizador. Se o valor inserido não for nenhum destes dois, então o programa inicia uma fase de finalização, onde removemos memória que estava a ser utilizada por arrays dinâmicos previamente definidos, e depois terminamos a execução do programa.

## Fila.h

O item slot.h é constituído por alguns *includes*, e pela definição de algumas estruturas e constantes. No que é relativo às constantes, as mais são *NUM\_PRODUTOS* e *NUM\_PRECOS*, que se referem respetivamente ao número de produtos e preços distintos encontrados nos ficheiros dados no enunciado do projeto (produtos.txt e precos.txt). A constante *EURO* é um *string* utilizado na impressão dos preços da máquina para a consola.

Esta máquina é um array de *slots*, estrutura que aqui se encontra definida, constituída por um produto, dois *inteiros* referentes à quantidade e capacidade (*quantidadeMax*), e um *char*, o código do slot. Cada produto está também aqui definido como armazenando um *string* para o seu nome e um *float* para o seu preço.

## Fila.cpp

As duas primeiras funções deste item, denominadas *lerFicheiroProdutos* e *lerFicheiroPrecos*, têm como objetivo ler respetivamente os ficheiros produtos.txt e precos.txt e retornar os arrays *listaprodutos* e *listaprecos* em que cada posição destes mesmos é uma linha do ficheiro correspondente. Estes arrays são utilizados para permitir que outras funções consigam trabalhar com os valores e strings presentes nos ficheiros.

A função *imprimeMaquina* tem como função imprimir a estrutura da máquina (imprimir todos os slots existentes na máquina e os seus conteúdos).

A função *precoDesteProduto* tem como funcionalidade a alteração do preço de um produto que existe em mais do que um slot da máquina. Caso houver mais do que um slot com o mesmo produto, os preços desses slots vão passar a ser iguais. Se o produto só existir em um só slot e ser novo, o preço desse produto vai ter um valor aleatório que está presente no ficheiro precos.txt.

A função *slotVazio* tem como objetivo retornar, num booleano, se a quantidade do slot é zero.

A função *VerificarVazio* vai percorrer a máquina toda e verifica *slotVazio*. Se retornar *true*, o nome do produto presente no slot muda para “Vazio” e o seu preço altera para 0,00.

A função *temVazio* é uma função booleana, que tem como objetivo verificar se o nome do produto é “Vazio”. Caso tal aconteça, a função irá retornar *true*.

A função *inicializaCodigo* só é usada na função em que é criada uma nova máquina. Esta função recebe um inteiro i, que é uma posição do slot da máquina, e retorna o código do slot que essa posição da máquina deverá ter.

A função *buscaCodigo* recebe um inteiro i, que representa a posição do slot da máquina e retorna o código do slot nessa posição da máquina.

A função *buscaPos* verifica de slot a slot se o código inserido na função é igual ao do slot da máquina. Se for true, ele retorna a posição do slot que contêm o código. Caso contrário, retorna -1.

A última função *createVendingMachine* cria uma nova máquina que contêm o número de slots *numSlots*, inicializado anteriormente como um valor aleatório entre 9 e 12. Cada slot vai conter um certo nome, preço e quantidade do produto, assim como a quantidade máxima/capacidade do slot e a letra (ou código) do slot. Durante essa implementação, é verificada em toda a máquina se o nome dos produtos nos slots anteriores são iguais ao nome do produto que será inserido no slot atual. Se for, o nome do produto será alterado e entra outra vez nesta verificação. Se o nome do produto for único, ou seja, não existir noutro slot que já esteja na máquina, então é introduzido no slot e avança para o slot seguinte, se houver. Depois da implementação, são imprimidos todos os slots existentes da máquina, usando a função *imprimeMaquina*.

## inicio.h

Este item é composto pelo protótipo das funções definidas em dinheiro.cpp, juntamente com alguns *includes*, como é habitual.

## inicio.cpp

Dentro deste ficheiro, temos várias funções referentes ao dinheiro e caixa de moedas.

A *função cria\_fundos* tem como objetivo criar os fundos (em moedas com valores de 2 euros, 1 euro, 50 cêntimos, 20 cêntimos, 10 cêntimos e 5 cêntimos) que serão utilizado como o troco da máquina.

Para armazenar a quantidade de moedas na máquina utilizamos o vetor dinâmico definido na inicialização, denominado de *moedas.* Cada elemento do vetor corresponde ao número de moedas de um certo valor, sendo que os elementos estão ordenados de forma decrescente do seu valor. Por exemplo, o primeiro elemento é o número de moedas de 2 euros existentes na máquina e o último elemento do vetor corresponde ao número de moedas de 5 cêntimos disponíveis. É calculado um valor aleatório entre 10 e 20 para cada elemento do vetor, e, por fim, há uma verificação dos fundos criados, através da função *verificarFundos*, que será explicada posteriormente,para se poder avisar o utilizador caso haja pouco troco na caixa de moedas, como segurança.

Esta função, *verificarFundos,* verifica se algum dos elementos do vetor *moedas* é menor ou igual a 3. Se esta condição for cumprida, o programa imprime uma mensagem de aviso que diz quais os tipos de moedas com pouco troco, alertando para a quantidade atual desses mesmos.

A função denominada *VerificarAviso* retorna um resultado booleano para esta mesma verificação (ou seja, *true* se houver um baixo número de algum tipo de moedas, ou *false* se não houver qualquer tipo de moedas que esteja com quantidade inferior a 4).

A função *imprimeFundos*imprime uma mensagem com o título *“\*\*\* Fundos \*\*\*”* , seguida dos valores armazenados para cada tipo de moedas, em ordem decrescente. O seu objetivo é imprimir os fundos atualmente disponíveis para que o utilizador saiba a quantidade de troco disponível na caixa de moedas em várias situações ao longo do decorrer do programa.

A função *remover\_fundos* primeiro mostra ao utilizador os fundos atuais da caixa de moedas, antes de passar por cada tipo de moeda e perguntar ao utilizador quantas desse tipo deseja remover. Se for inserido um valor não válido, ou seja, que cause com que o número final de moedas fique negativo, é pedido ao utilizador que reintroduza o número de fundos desse tipo a remover. Após passar por todos os tipos de moedas, é imprimida a caixa de moedas atualizada com a remoção feita pelo funcionário.

A função *carregar\_moedas* tem por objetivo carregar as moedas da máquina, ao adicionar um certo valor desejado ao valor inicial presente nessa posição do vetor referente à caixa de moedas.

## cliente.h

É constituído pelo protótipo da função *menu\_cliente* e alguns #includes referentes a esta mesma.

## cliente.cpp

Neste, existe apenas uma função, *menu\_cliente*, que é referente ao menu que o cliente irá aceder. Neste menu, é pedido para o cliente inserir o código do produto desejado. Se o mesmo não for encontrado na máquina atual, é pedido que seja inserido um código válido. Após escolhido um produto, o programa pede sequencialmente a quantidade de moedas de cada tipo que o cliente deseja pagar com, e depois são feitas várias verificações:

Se o valor inserido em euros for inferior ao custo do produto, a transação termina, o sistema devolve o troco, e retorna-se ao menu anterior. Se o slot escolhido estiver vazio, ou seja, se a quantidade nesse slot for 0, então a transação também termina da mesma forma que na condição anterior. Se a condição *VerificarAviso*, explicada anteriormente, se verificar, a transação também termina. Noutro caso que não estes últimos, então a transação continua.

No próximo momento da transação, se o valor inserido pelo cliente for superior ao valor referente ao produto que pediu, são contadas sequencialmente o número de moedas de cada tipo que a máquina irá devolver, em ordem decrescente dos seus valores. Depois, dando-se por completa a transação, devolve o produto e o troco ao utilizador, imprime os fundos atuais da máquina e por fim verifica se algum dos tipos de moedas da caixa se encontra inferior ou igual a 3. Caso tal acontecer, irá avisar o utilizador com uma mensagem de alerta.

## funcionario.h

São definidos aqui os protótipos das funções de *funcionário.cpp,* juntamente com uma referência à variável global *numSlots* para que possa ser atualizada e reconhecida pelas funções *carregarMaquinanumSlots* e *adicionarSlot*.

## funcionario.cpp

A função *menu\_funcionário* é o menu principal para o funcionário efetuar operações sobre a máquina, permitido que escolha entre 11 opções, as quais estão definidas ao longo deste mesmo ficheiro.

A primeira função *imprimirProdutos* permite que o funcionário escolha 3 opções: imprimir os produtos por ordem alfabética, por preço ou por quantidade disponível. São declarados 3 vetores dinâmicos, um denominado *precos*, do tipo *float*, um chamado *produtos*, do tipo *string* e por último o *quantidade*, do tipo *int* que permite a ordenação destes. Depois é utilizado o *bubblesort* de acordo com a escolha do utilizador para percorrer o vetor, ordená-lo, e depois imprimir o resultado desejado. Por fim, eliminam-se os 3 vetores dinâmicos utilizados.

A segunda função é o *alterar\_preco.* Nesta função o utilizador introduz o código do produto que deseja alterar. Caso o produto não seja encontrado, o programa repetirá a pergunta até o utilizador introduzir um produto existente na máquina. Quando encontra o produto, o programa permite ao utilizador alterar o preço dele, desde que respeite as condições de ser positivo e múltiplo de cinco. Se o preço inserido não as respeitar, é pedido um novo preço. Após receber um preço aceitável, o programa altera o preço do slot atual, e depois de todos os outros slots cujo nome do produto é o mesmo. Assim, dois slots com o mesmo produto terão sempre o mesmo preço.

Nas funções *gravarMaquinaSlots*, *gravarMaquinanumSlots* e *gravarMaquinaMoedas,*  usufrui-se do *fstream* para que seja possível abrir o ficheiro e consequentemente gravar respetivamente os slots (que involvem o nome, preço, código, quantidade e capacidade), o valor de *numSlots* (que é o tamanho da máquina atual), e o número de moedas de cada tipo (que estão no vetor *moedas*), nos ficheiros passados como argumento de cada função. Caso não se consiga abrir algum ficheiro, irá surgir uma mensagem de erro na consola. Cada função deverá gravar os conteúdos num ficheiro distinto.

As funções *carregarMaquinanumSlot* , *carregarMaquinaMoedas* e *carregarMaquinaSlots* ambas usam o *fstream* e o *getLine* para ler e também invocam uma variável *line* para que seja armazenada as informações contidas no ficheiro. As funções *carregarMaquinaMoedas* e *carregarMaquinaSlots* usam um vetor dinâmico do tipo *char* que auxiliam na leitura.

Na função *limparSlot*, é pedido ao utilizador o slot que deseja limpar. Caso não encontre um slot com o mesmo código na máquina, o programa torna a pedir um código, só aceitando um que exista na máquina. Se for válido, retira os produtos deste slot e altera o seu nome para “Vazio” e o preço para 0,00€, mas mantém a capacidade máxima desse slot.

A função *limparMaquina* percorre todos os slots existentes na máquina e limpa-os da mesma forma que a função *limparSlot*, mas não pede ao utilizador que introduza qualquer código, visto tal não ser necessário.

A função *adicionarProdutos* permite o utilizador introduzir o código de um slot para atualizar a quantidade de produtos nele, ou até adicionar um novo produto num slot que esteja marcado como “Vazio”. Este novo produto pode ter qualquer nome. O preço do produto será igual a outro já na máquina se existir. Se o produto inserido não existir anteriormente na máquina, é lhe dado um novo valor aleatório. Esta verificação e execução é levada a cabo pela função *precoDesteProduto*.

Caso a quantidade inserida seja superior à capacidade do slot, é perguntado ao utilizador se deseja alterar o tamanho do slot para que caibam os produtos adicionados, adicionar apenas até à capacidade máxima do slot, ou, se possível, inserir a quantidade de produtos a mais num slot que esteja vazio. Segundo a escolha do utilizador, é efetuada a operação correspondente.

# Conclusão

Neste trabalho, conseguimos cumprir todos os objetivos que nos tinham sido propostos no enunciado. Tentámos fazer a máquina com base na melhor eficiência que nos era possível. Poderíamos ter usado um algoritmo de ordenação mais eficiente que o bubble sort, na função *imprimirProdutos*, mas mesmo assim, nós achamos que, para a sua complexidade, este algoritmo de ordenação era suficientemente bom para o objetivo desta função.

Relativamente à utilização de uma variável global, apesar de usualmente não ser o mais aconselhável, achámos que seria a melhor e mais simples forma de adaptar o nosso código a permitir que o valor *numSlots*, presente na função main.cpp, fosse o mesmo que o local criado no interior do *menu\_funcionário*, pois não se tratava de uma constante. Na ausência da variável global, a alteração do inteiro dentro do *menu\_funcionário* não conseguia atualizar o valor guardado deste inteiro do mesmo nome na função *main*, e foi por isso que decidimos prosseguir com uma variável global que pudesse dessa forma ser atualizada.

A máquina em si é inicializada com 100 slots, para ter certeza que não ocorrerão problemas na execução normal do programa, mas isto não deixa de ser algo relativamente ineficiente, em termos de memória alocada. Contudo, decidimos que seria o modo mais simples de permitir a adição de slots e carregamento fluido de máquinas guardadas, tendo em conta que não podíamos utilizar estruturas como as listas ligadas.

Abaixo encontram-se vários links que usámos como referência para algumas partes do nosso código:

const\_cast<char\*> (const char\*) :

[(C++) const char\* to char\* - how? - For Beginners](https://www.gamedev.net/forums/topic/495664-c-const-char-to-char-how/)

cout << fixed << setprecision(2) :

[http://www.cplusplus.com/reference/iomanip/setprecision](http://www.cplusplus.com/reference/iomanip/setprecision/)

variável global (extern int numSlots) :

[How to declare a global variable in C++](https://stackoverflow.com/questions/9702053/how-to-declare-a-global-variable-in-c)

[Scope of Variables in C++](https://www.geeksforgeeks.org/scope-of-variables-in-c/)