**Introdução**

No âmbito da disciplina de EDA, foi-me proposto a realização de um projeto prático, projeto este que consiste no desenvolvimento de uma empresa de autocarros em C++. Para a realização eficiente deste projeto, foram criadas várias estruturas de forma a gravar os dados necessários em memória, e manter a leitura do código fácil. Este programa contém também várias funções com o objetivo de abordar os vários problemas que me 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, e mostrar o seu funcionamento.

É também de notar que algum código se encontra comentado e marcado como DEBUG. Este código foi deixado de maneira a tornar algum futuro debug do programa mais rápido, e ajudar a compreender o que o código está a fazer, por parte de quem o estiver a executar.

**Pré-Inicialização**

**Locale, consola do Windows e Encodings**

Tendo em conta que os ficheiros de inicialização contêm nomes portugueses, e de forma a possibilitar o uso de carateres especiais tanto nas mensagens imprimidas na consola, quanto no input por parte do utilizador, as seguintes linhas de código são usadas.



Referências:

<https://docs.microsoft.com/en-us/windows/console/setconsoleoutputcp?redirectedfrom=MSDN>

**Geração de números pseudoaleatórios**

Para tornar possível a geração de números pseudoaleatórios 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 aleatoriamente na inicialização do programa o número de paragens. Mais tarde a função será também usada para escolher aleatoriamente os nomes das paragens, passageiros e motoristas, assim como a capacidade de cada autocarro.

Adivinhar estas caraterísticas não apresenta nenhum problema de segurança ao programa, por isso não é necessária uma implementação mais complexa que permita gerar números mais difíceis de adivinhar.



Referências:

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

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

**Ficheiros de inicialização**

Para o programa ser inicializado corretamente, foi imposto que esta retirasse os vários nomes de ficheiros correspondentes. Foi deixado ao utilizador alterar o código fonte do programa e escolher a localização destes ficheiros.

**Abertura dos ficheiros de inicialização**

Após o utilizador inserir a localização dos três ficheiros, estes ficheiros têm que ser abertos pelo programa, e todos os nomes de lá retirados. Existem várias formas de abordar o problema, nomeadamente:

* Abrir 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 dispositivo 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 nome ocupa 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.

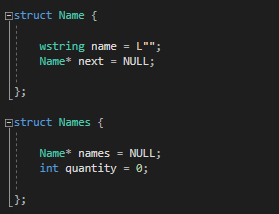
**Alocação de memória para gravação dos ficheiros**

Para guardar os documentos na memória RAM foram criadas:

* Uma função, com o objetivo de fazer “*parsing*” aos documentos de inicialização.



* Duas estruturas:
  + Uma lista ligada que guarda todos os nomes do documento escolhido. Foi escolhida uma lista ligada de forma a permitir ir alocando memória e adicionando os nomes à mesma conforme estes vão sendo lidos no documento. Este comportamento não seria possível usando um simples array.
  + Uma estrutura que contém a lista ligada anterior e o número de nomes que a mesma contém.



**Erros ao abrir/“parsing” dos ficheiros**

Se por alguma razão algum dos ficheiros de inicialização não forem abertos corretamente, o seguinte erro é mostrado, e o programa não é inicializado.



Se por alguma razão algum dos ficheiros de inicialização não forem abertos corretamente, ou algum dos ficheiros não conter nenhum nome, o seguinte erro é mostrado, e o programa não é inicializado.

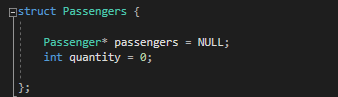


**Inicialização do programa**

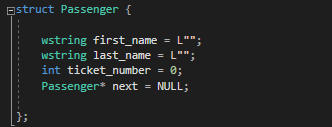
Se todos os ficheiros forem abertos corretamente, ambos os ficheiros dos nomes próprios conterem pelo menos um nome, e o ficheiro dos nomes das paragens conter nomes suficientes de forma a permitir uma inicialização correta o programa é inicializado.

**Alocação de memória para a lista de espera**

A lista de espera consiste numa estrutura que contém uma lista ligada de passageiros (outra estrutura) e o número de passageiros nela presentes.



Cada passageiro é caraterizado pelo seu primeiro e último nome, assim como o seu número do bilhete (número este único a cada passageiro). Cada passageiro guarda também o endereço de memória do passageiro que se encontra a seguir na fila.



**Inicialização da fila de espera**

Esta inicialização é realizada pela função *initialize\_queue*. É gerada uma lista ligada de que contém 30 passageiros. Este processo consiste em alocar memória para cada passageiro, sendo este logo de seguida inicializado.

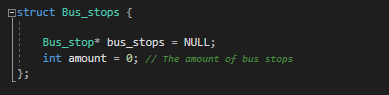


A função usada para inicializar cada passageiro é a função *initialize\_passenger.*

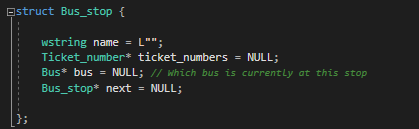


**Alocação de memória para as paragens**

Para as paragens foi criada uma estrutura que contém uma lista ligada de paragens (outra estrutura) e o número de paragens nela presentes.



Cada paragem é caraterizada pelo nome, assim como o autocarro nela presente (se existir algum), e uma árvore de procura binária que contém todos os números dos bilhetes dos passageiros que nela saíram. Cada paragem guarda também o endereço de memória da próxima paragem da lista ligada.



**Inicialização das paragens**

Esta inicialização é realizada pela função *initialize\_bus\_stops*.



É gerado um número pseudoaleatório entre 4 e 9 (incluindo ambos) que será igual ao número de paragens a serem inicializadas.



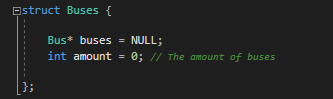
Logo de seguida é gerada uma lista ligada de que contém um número de paragens igual ao número aleatório que acabou de ser gerado. Este processo consiste em alocar memória para cada paragem, sendo esta imediatamente inicializada.

A função usada para inicializar cada paragem é a função *initialize\_bus\_stop*.



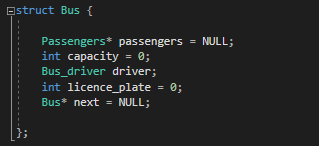
**Alocação de memória para os autocarros**

Para os autocarros foi criada uma estrutura que contém uma lista ligada de autocarros (outra estrutura) e o número de autocarros nela presentes.

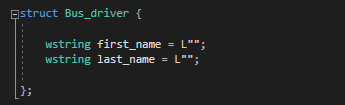


Cada autocarro é caraterizado pela sua matrícula (guardada como inteiro, mas sendo imprimida como hexadecimal), pelo seu condutor (outra estrutura), pela sua capacidade e pelos passageiros e número de passageiros que nele se encontram. Cada autocarro guarda também o endereço de memória do próximo autocarro da lista ligada.

É de notar que ao usar imprimir hexadecimais, usa-se *hex* antes da impressão dos mesmos, sendo logo de seguida necessário o uso do *dec.*

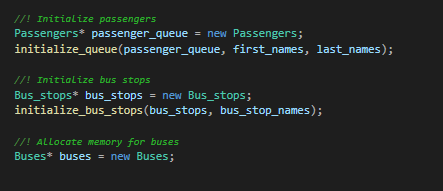


Cada condutor é caraterizado pelo seu primeiro e último nome.



**Inicialização dos autocarros**

Quando o programa começa são criadas e inicializadas a lista de espera a lista ligada de paragens. O mesmo não acontece com os autocarros, que apenas têm reservados em memória o espaço necessário para o primeiro autocarro, ainda não inicializado.



Cada autocarro será inicializado mais tarde, conforme vão sendo adicionados. Esta inicialização é realizada pela função *initialize\_bus*.



**Interface do programa**

É de salientar que algumas mensagens de erro sofreram pequenas alterações de forma a melhorar a experiência do utilizador, podendo não se encontrar exatamente como no relatório.

**Função *refresh\_console***

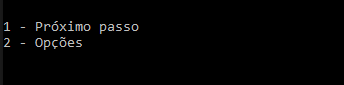
Esta é a função encarregue de limpar a consola e imprimir ao utilizador todos os dados necessários para uma fácil e rápida interação com a máquina, entre estes a lista de espera, as paragens, e os autocarros.

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*.



**Menu Principal**

Se a inicialização da máquina decorrer sem nenhum problema o programa abre o seu menu principal, onde o utilizador dirá ao programa se deseja passar para o passo seguinte, ou aceder as suas opções.



**Próximo passo**

Se o utilizador escolher a opção “Próximo passo” no menu principal acontece o seguinte.

São adicionados 15 passageiros à fila de espera. Se estes passageiros fossem adicionados após a criação do autocarro poderiam surgir problemas (nomeadamente inicializar um autocarro com uma fila de espera vazia), como tal eu implementei a solução de forma a permitir ambos os casos, de forma a permitir esta mudança caso futuramente seja desejável.

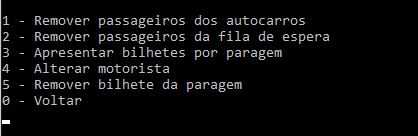
Todos os autocarros existentes são movidos para a próxima paragem.

É adicionado um autocarro à primeira paragem.

Cada passageiro tem uma probabilidade de sair do autocarro na paragem onde este se encontra, e ter o seu número do bilhete adicionado à paragem.

**Opções**

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



**Remover passageiros dos autocarros**

Abre um menu onde o utilizador escolhe o nº do passageiro que deseja remover dos autocarros.



Se não existirem autocarros, mostra a seguinte mensagem.



Se existirem autocarros, mas já todos estiverem vazios, mostra a seguinte mensagem.



Se não existir nenhum passageiro com o número inserido em nenhum autocarro, mostra a seguinte mensagem.



Se o passageiro for removido dos autocarros com sucesso, mostra a seguinte mensagem.



**Remover passageiros da fila de espera**

Abre um menu onde o utilizador escolhe o nº do passageiro que deseja remover da fila de espera.



Se a fila de espera estiver vazia, mas já todos estiverem vazios, mostra a seguinte mensagem.



Se não existir nenhum passageiro com o número inserido, mostra a seguinte mensagem.



Se o passageiro for removido da lista de espera com sucesso, mostra a seguinte mensagem.



**Apresentar bilhetes por paragem**

Abre um menu onde o utilizador escolhe o nome da passagem que contém os bilhetes que deseja apresentar.

Se ainda não existirem autocarros



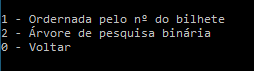
Se a passagem não tiver passageiros:



Se a paragem escolhida não existir:



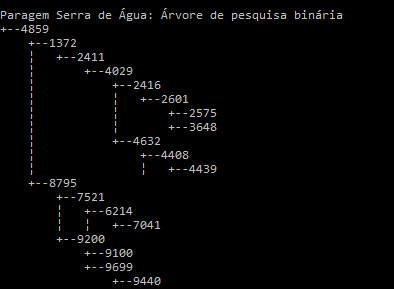
Se a paragem existir e tiver passageiros, mostra o seguinte menu:



Se o utilizador inserir o número 1 são imprimidos todos os números dos bilhetes por ordem crescente:



Se o utilizador inserir o número 2 são imprimidos todos os números dos bilhetes na forma de árvore binária:



Referências:

<https://stackoverflow.com/questions/36802354/print-binary-tree-in-a-pretty-way-using-c>

Se o utilizado inserir o número 0 volta ao menu das opções.

**Alterar Motorista**

Abre um menu onde o funcionário escolhe o nº da matrícula do autocarro cujo motorista deseja mudar.



Se não existirem autocarros, mostra a seguinte mensagem.



Se não existir nenhum autocarro com a matrícula escolhida, mostra a seguinte mensagem.



Se a matrícula inserida existir, pede ao utilizador que insira o primeiro e último nome do novo motorista.



**Remover bilhete da paragem**

Abre um menu onde o funcionário escolhe o nº do passageiro que deseja remover da paragem. Os vários “checks” realizados para o fácil funcionamento por parte do utilizador são iguais aos realizados ao “Apresentar bilhetes por paragem”.

Se a paragem existir e tiver passageiros, mostra a seguinte mensagem:



Se o número inserido não existir, mostra a seguinte mensagem:



Se o número inserido existir, o número do passageiro é removido por fusão, de forma a manter a árvore o mais equilibrada possível, e mostra a seguinte mensagem:



**Voltar**

Volta ao menu principal.

**Possíveis melhorias**

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

* Os ficheiros dos nomes poderiam sofrer uma validação, de forma a não conter números, ou nomes repetidos.
* Realizar um “check” que verifica que em que sistema operativo o programa está a correr, e muda os comandos encontrados no programa de acordo.
* Realizar um “check” que verifica se o documento dos nomes das paragens contém nomes suficientes para uma boa execução do programa.

**Conclusão**

Com base no conhecimento obtido nas aulas gravadas pelo professor, foi possível realizar este segundo projeto com sucesso. O desenvolvimento do projeto obrigou-me a aprofundar conhecimentos acerca de estruturas de dados e algoritmos, entre estes as listas ligadas e as árvores de procura binária e consequente remoção, adição, e procura de nós nas mesmas. Este projeto foi mais interessante de realizar do que o primeiro, por ser necessário um maior nível de abstração em relação ao problema exigido. Sem dúvida alguma que estou mais bem preparado para lidar com problemas futuros que possam surgir após a sua conclusão, pois antes da realização deste projeto nunca tinha tido interesse pelo uso do software git, nem pensado em árvores como uma implementação para resolver um problema.