iscte TECNOLOGIAS E ARQUITETURA Departamento de Ciências e Tecnologias da Informação	Sistemas Operativos Ano Letivo 2023/2024 2º Semestre Parte 1 (Versão 2)

Projeto *IscteFlight* (Parte 1)

O presente trabalho visa aplicar os conhecimentos adquiridos durante as aulas de Sistemas Operativos e será composto por três partes. O objetivo é desenvolver os diferentes aspetos e funcionalidades da plataforma **IscteFlight**. Iremos procurar minimizar as interdependências entre partes do trabalho.

Este enunciado detalha **apenas** as funcionalidades que devem ser implementadas na parte 1 do trabalho.
Este trabalho está publicado na sua **versão 2**. Consulte o Histórico de Versões para ver as alterações efetuadas.

A plataforma **IscteFlight** é um sistema de gestão de reservas de voos totalmente automatizada. O seu objetivo é facilitar a interação entre os passageiros e os serviços oferecidos por uma companhia aérea. Na aplicação **IscteFlight** existem os seguintes conceitos:

Passageiro: pessoa que reserva o voo. Os passageiros devem estar registados na plataforma **IscteFlight** com os dados: *ID Passageiro*, *NIF* (Nº Contribuinte), *Nome*, *Email*, *Senha* (password), *Saldo* (para compra de viagens).

Voo: a viagem que está disponível para reserva. Os voos são caracterizados por: *Nº Voo*, *Origem*, *Destino*, *Data*, *Hora*, *Preço*, *Lotação*, *Nº Lugares Disponíveis*.

Relatório de reservas: relatório emitido com a descrição das reservas de voos registadas no sistema, contendo: *ID Reserva*, *Nº Voo*, *Origem*, *Destino*, *ID Passageiro*, *Preço*, *Data* e *Hora da reserva*.

Em termos gerais, um novo passageiro regista-se na aplicação **IscteFlight** fornecendo os seus dados. Ao efetuar o registo, o *passageiro* deve carregar o seu saldo com créditos. Após o registo, o *passageiro* poderá reservar *voos* quando estes estiverem disponíveis. No ato da reserva, o número de lugares disponíveis é decrementado, é descontado o preço do bilhete do saldo do passageiro e é adicionada uma linha no *relatório de reservas*. Diariamente, de hora em hora, um procedimento automático faz a atualização dos estados dos *voos*.

Os alunos não deverão usar o programa **`echo`** (não será analisado para efeitos de avaliação, a não ser que seja explicitamente indicado) para as respostas diretas às alíneas. Deverá sim usar as macros **`so_success`** (para as respostas de sucesso) e **`so_error`** (para as respostas de insucesso), ver exemplos de uso no final deste enunciado. O programa **`echo`** pode, no entanto, ser usado para as interações com o utilizador, ou para escrever em ficheiros.

A *baseline* para o trabalho encontra-se no Tigre, na diretoria **`/home/so/trabalho-2023-2024/parte-1`**:

- **EXECUTE**, **a partir da sua diretoria local de projeto**, e sem mudar os nomes, o seguinte comando:

```
$ cp -r /home/so/trabalho-2023-2024/parte-1  .
```

Procedimento de entrega e submissão do trabalho

O trabalho de SO será realizado **individualmente**, logo sem recurso a grupos.

A entrega da Parte 1 do trabalho será realizada através da criação de **um** ficheiro ZIP cujo nome é o nº do aluno, e.g., "**a<nºaluno>-parte-1.zip**" (**ATENÇÃO:** não serão aceites ficheiros RAR, 7Z ou outro formato) onde estarão todos os ficheiros criados. Estes serão **apenas** os ficheiros de código, ou seja, na parte 1, apenas os ficheiros (***.sh *.def**).

Cada um dos módulos será desenvolvido com base nos ficheiros fornecidos, e que estão na diretoria do Tigre **"/home/so/trabalho-2023-2024/parte-1"**, e deverá incluir nos comentários iniciais um "relatório" indicando a descrição do módulo e explicação do mesmo (poderá ser muito reduzida se o código tiver comentários descritivos).

Para criar o ficheiro ZIP, use, no Tigre, o comando **zip a<nº aluno>-parte-1.zip <ficheiros>**, por exemplo:

```
$ zip $USER-parte-1.zip *.sh *.def
```

O ficheiro ZIP deverá depois ser transferido do Tigre para a sua área local (Windows/Linux/Mac) via SFTP, para depois ser submetido via Moodle.

Antes de submeter, por favor valide que o ficheiro ZIP não inclui diretorias ou ficheiros extra indesejados.

A entrega desta parte do trabalho deverá ser feita por via eletrónica, através do Moodle:

- Moodle da UC Sistemas Operativos, selecione a opção sub-menu "Quizzes & Assignments".
- Selecione o link "Submit SO Assignment 2023-2024 Part 1".
- Dentro do formulário, selecione o botão "Enviar trabalho" e anexe o seu ficheiro **.zip** (a forma mais fácil é simplesmente fazer via "drag-and-drop") e selecione o botão "Guardar alterações". Poderá depois, mais tarde, ressubmeter o seu trabalho as vezes que desejar, enquanto estiver dentro do prazo para entrega do trabalho. Para isso, na mesma opção, pressione o botão "Editar submissão", selecione o ficheiro, e depois o botão "Apagar", sendo que depois pode arrastar o novo ficheiro **.zip** e pressionar "Guardar alterações". **Apenas a última submissão será contabilizada**. Certifique-se que a submissão foi concluída, e que esta última versão tem todas as alterações que deseja entregar, dado que os docentes apenas considerarão esta última submissão.
- Avisamos que a hora *deadline* acontece sempre poucos **minutos antes da meia-noite**, pelo que se urge a que os alunos não esperem por essa hora final para entregar e o façam antes, idealmente um dia antes, ou no pior dos casos, pelo menos uma hora antes. **Não serão consideradas válidas as entregas realizadas por e-mail**. Poderão testar a entrega nos dias anteriores para perceber se há algum problema com a entrega, sendo que, **apenas a última submissão conta**.

Política em caso de fraude

O trabalho corresponde ao esforço individual de cada aluno. São consideradas fraudes as seguintes situações: Trabalho parcialmente copiado, facilitar a cópia através da partilha de ficheiros, ou utilizar material alheio sem referir a fonte.

Em caso de deteção de fraude, os trabalhos em questão não serão avaliados, sendo enviados à Comissão Pedagógica da escola (ISTA) ou ao Conselho Pedagógico do ISCTE, consoante a gravidade da situação, que decidirão a sanção a aplicar aos alunos envolvidos. Serão utilizadas as ferramentas *Moss* e *SafeAssign* para deteção automática de cópias.

Recorda-se ainda que o Anexo I do Código de Conduta Académica, publicado a 25 de janeiro de 2016 em Diário da República, 2ª Série, nº 16, indica no seu ponto 2 que quando um trabalho ou outro elemento de avaliação apresentar um nível de coincidência elevado com outros trabalhos (percentagem de coincidência com outras fontes reportada no relatório que o referido software produz), cabe ao docente da UC, orientador ou a qualquer elemento do júri, após a análise qualitativa desse relatório, e em caso de se confirmar a suspeita de plágio, desencadear o respetivo procedimento disciplinar, de acordo com o Regulamento Disciplinar de Discentes do ISCTE - Instituto Universitário de Lisboa, aprovado pela deliberação nº 2246/2010, de 6 de dezembro.

O ponto 2.1 desse mesmo anexo indica ainda que no âmbito do Regulamento Disciplinar de Discentes do ISCTE-IUL, são definidas as sanções disciplinares aplicáveis e os seus efeitos, podendo estas variar entre a advertência e a interdição da frequência de atividades escolares no ISCTE-IUL até cinco anos.

Parte I – Shell Script (bash)

Data de entrega: 10 de março de 2024

Nesta fase do trabalho, o objetivo é criar um conjunto de scripts para administração e gestão da plataforma.

Atenção: Apesar dos vários ficheiros necessários para a realização do trabalho serem fornecidos na diretoria do Tigre "**/home/so/trabalho-2023-2024/parte-1**", assume-se que, para a sua execução, os scripts e todos os ficheiros de input e de output estarão todos **sempre** presentes na mesma diretoria, que não deve estar *hard-coded*, ou seja, os Shell scripts entregues devem correr em qualquer diretoria.

S1. Script: `regista_passageiro.sh`

Este script é invocado quando um novo passageiro se regista na plataforma **IscteFlight.** Este script recebe todos os dados por argumento, na chamada da linha de comandos. Os passageiros são registados no ficheiro **`passageiros.txt`**. Deve receber as informações do passageiro como argumentos pela seguinte ordem:

```
<Nome:string> <Senha:string>  <Saldo a adicionar:number> [<NIF:number>]
```

Exemplos de invocação deste script, que deverá receber os valores pedidos passados como argumentos:

```
$ ./regista_passageiro.sh "David Gabriel" 12qwaszx 10
$ ./regista_passageiro.sh " Nuno Garrido" 09polkmn 20 213654377
```

Exemplo do ficheiro **`passageiros.txt`**:

Passos do script:

S1.1. Valida os argumentos passados e os seus formatos:

S1.1.1. Valida os argumentos passados, avaliando se são em número suficiente (mínimo 3, máximo 4). Em caso de erro, dá **`so_error S1.1.1`** e termina. Caso contrário, dá **`so_success S1.1.1`**.

S1.1.2. Valida se o argumento **<Nome>** corresponde ao nome de um utilizador do servidor Tigre. Se não corresponder ao nome de nenhum utilizador do Tigre, dá **`so_error S1.1.2`** e termina. Senão, dá **`so_success S1.1.2`**.

S1.1.3. Valida se o argumento **<Saldo a adicionar>** tem formato "*number*" (inteiro positivo ou 0). Se não tiver, dá **`so_error S1.1.3`** e termina. Caso contrário, dá **`so_success S1.1.3`**.

S1.1.4. Valida se o argumento opcional **<NIF>** (só no caso de ser passado, i.e., se tiver valor) tem formato "*number*" com 9 (nove) dígitos. Se não for, dá **`so_error S1.1.4`** e termina. Caso contrário, dá **`so_success S1.1.4`**.

S1.2. Associa os dados passados com a base de dados dos passageiros registados:

S1.2.1. Verifica se o ficheiro **`passageiros.txt`** existe. Se o ficheiro existir, dá **`so_success S1.2.1`** e continua no passo **S1.2.3**. Se não existir, dá **`so_error S1.2.1`**, e continua.

S1.2.2. Cria o ficheiro **`passageiros.txt`**. Se der erro, dá **`so_error S1.2.2`** e termina. Senão, dá **`so_success S1.2.2`**.

S1.2.3. Caso o passageiro <Nome> passado já exista no ficheiro `passageiros.txt`, dá **`so_success S1.2.3`**, e continua no passo **S1.3**. Senão, dá **`so_error S1.2.3`**, e continua.

S1.2.4. Como o passageiro **<Nome>** não existe no ficheiro, terá de o registar. Para isso, valida se **<NIF>** (campo opcional) foi mesmo passado. Se não foi, dá **so_error S1.2.4** e termina. Senão, dá **so_success S1.2.4**.

S1.2.5. Define o campo **<ID_passageiro>**, como sendo o UserId Linux associado ao utilizador de nome **<Nome>** no servidor Tigre. Em caso de haver algum erro na operação, dá **so_error S1.2.5** e termina. Caso contrário, dá **so_success S1.2.5 <ID_passageiro>** (substituindo pelo campo definido).

S1.2.6. Define o campo **<Email>**, gerado a partir do **<Nome>** introduzido pelo utilizador, usando apenas o primeiro e o último nome, convertendo-os para minúsculas apenas, colocando um ponto entre os dois nomes, e domínio `iscteflight.pt`. Assim sendo, um exemplo seria "`david.gabriel@iscteflight.pt`". Se houver algum erro na operação (e.g., o utilizador "`root`" tem menos de 2 nomes), dá **so_error S1.2.6** e termina. Caso contrário, dá **so_success S1.2.6 <Email>** (substituindo pelo campo gerado). Ao registar um novo passageiro no sistema, o número inicial de **<Saldo>** tem o valor `0` (zero).

S1.2.7. Regista o utilizador numa nova linha no final do ficheiro **passageiros.txt**, seguindo a sintaxe: **<ID_passageiro>:<NIF>:<Nome>:<Email>:<Senha>:<Saldo>**. Em caso de haver algum erro na operação (e.g., erro na escrita do ficheiro), dá **so_error S1.2.7** e termina. Caso contrário, dá **so_success S1.2.7 <linha>** (substituindo pela linha completa escrita no ficheiro).

S1.3. Adiciona créditos na conta de um passageiro que existe no ficheiro **passageiros.txt**:

S1.3.1. Tendo já encontrado um "match" passageiro com o Nome **<Nome>** no ficheiro, valida se o campo **<Senha>** passado corresponde à senha registada no ficheiro. Se não corresponder, dá **so_error S1.3.1** e termina. Caso contrário, dá **so_success S1.3.1**.

S1.3.2. Mesmo que tenha sido passado um campo **<NIF>** (opcional), ignora-o. Adiciona o valor passado do campo **<Saldo a adicionar>** ao valor do **<Saldo>** registado no ficheiro **passageiros.txt** para o passageiro em questão, atualizando esse valor no ficheiro **passageiros.txt**. Se houver algum erro na operação (e.g., erro na escrita do ficheiro), dá **so_error S1.3.2** e termina. Caso tudo tenha corrido bem, dá o resultado **so_success S1.3.2 <Saldo>** (substituindo pelo valor saldo atualizado no ficheiro **passageiros.txt**).

S1.4. Lista todos os passageiros registados, mas ordenados por saldo:

S1.4.1. O script deve criar um ficheiro chamado **passageiros-saldos-ordenados.txt** igual ao que está no ficheiro **passageiros.txt**, com a mesma formatação, mas com os registos ordenados por ordem decrescente do campo **<Saldo>** dos passageiros. Se houver algum erro (e.g., erro na leitura ou escrita do ficheiro), dá **so_error S1.4.1**, e termina. Caso contrário, dá **so_success S1.4.1**.

NOTA: Este script **S1** foi descrito da forma mais explícita possível, para que os alunos percebam a forma como se pretende que as perguntas deste trabalho de SO sejam respondidas, ou seja, **que TODOS os passos de cada script de todos os scripts (até mesmo os seguintes, onde, para reduzir a complexidade do enunciado, apenas será dito para dar so_success ou so_error)** tenham sempre uma resposta com formato **so_success <passo>**, e, na maioria das vezes, também uma resposta de formato **so_error <passo>**, sendo que, por vezes, a mensagem de erro implica terminar completamente o script, e noutras o comportamento esperado será continuar o script (este comportamento será especificado caso a caso). Para a validação de cada passo, é essencial que as respostas tenham o formato especificado. Em alguns casos, poderá ser especificado que para além da mensagem de sucesso ou erro, seja passada mais informação, como é o caso da pedida em **S1.2.5**, entre outros. Nesses casos, estas macros devem ser invocadas usando a sintaxe **so_success <passo> <informação1> ... <informaçãoN>** ou **so_error <passo> <informação1> ... <informaçãoN>**. Os alunos poderão, nos seus scripts, ter outros outputs realizados com **echo** ou outras formas, mas tais outputs não serão contabilizados, a não ser que seja explicitamente indicado.

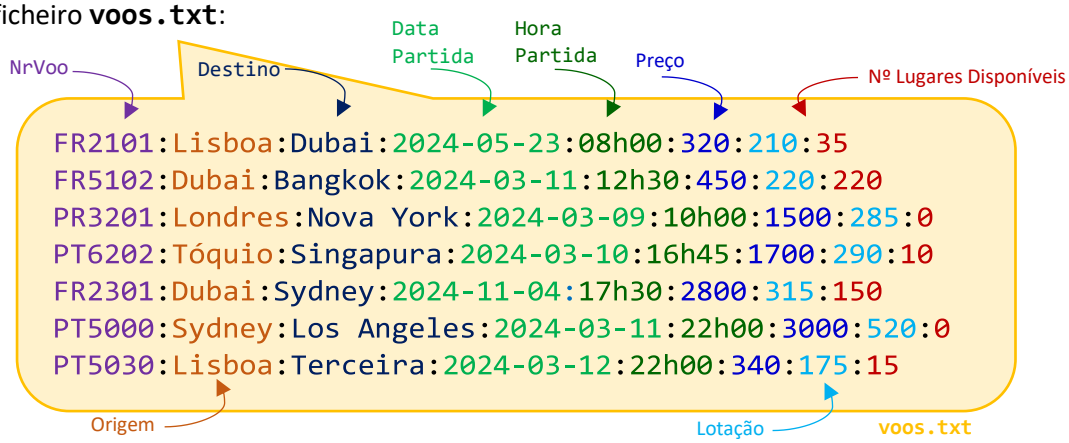
S2. Script: **compra_bilhete.sh**

Este script não recebe nenhum argumento, e permite que o passageiro compre um bilhete para um voo da lista de voos disponíveis. Para realizar a compra, o passageiro deve fazer login para confirmar sua identidade e saldo disponível. Os voos disponíveis estão listados no ficheiro **voos.txt**.

O ficheiro **voos.txt** tem a informação estática sobre as características do voo, com o seguinte formato:

<NrVoo:string>:<Origem:string>:<Destino:string>:<DataPartida:string>:<HoraPartida:string>:<Preço: number>:<Lotação:number>:<Lugares Disponíveis:number>

Exemplo do ficheiro **voos.txt**:



Sendo:

<Data Partida>: Data em que se realizará o voo, no formato AAAA-MM-DD.
<Hora Partida>: Hora em que se realizará o voo, no formato HHhMM.
<Preço>: O preço do bilhete do voo (inteiro positivo ou 0, não tem nunca um valor decimal).
<Lotação>: Nº máximo de passageiros (lugares) que o avião pode transportar.
<Lugares Disponíveis>: Nº de lugares disponíveis no voo.

S2.1. Validações e Pedido de informações interativo:

S2.1.1. O script valida se os ficheiros **voos.txt** e **passageiros.txt** existem. Se algum não existir, dá **so_error** e termina. Caso contrário, dá **so_success**.

S2.1.2. Na plataforma é possível consultar os voos pela sua **<Origem>** ou **<Destino>**. Pedindo:

```
Insira a cidade de origem ou destino do voo: _
```

O utilizador insere a cidade Origem ou Destino do voo (o interesse é que pesquise nos 2 campos). Caso o utilizador tenha introduzido uma cidade que não exista no ficheiro **voos.txt**, ou se não existirem voos com lugares disponíveis com origem ou destino nessa cidade, dá **so_error** e termina. Caso contrário, dá **so_success <Cidade>**.

S2.1.3. O programa pede ao utilizador para inserir uma opção de voo, listando os voos que existem de acordo com a origem/destino inserida anteriormente, da seguinte forma:

```
1.Lisboa para Dubai, 2024-05-23, Partida:08h00, Preço: 320, Disponíveis:35 lugares
2.Dubai para Bangkok, 2024-03-11, Partida:12h30, Preço: 450, Disponíveis:220 lugares
3.Dubai para Sidney, 2024-11-04, Partida:17h30, Preço: 2800, Disponíveis:150 lugares
0.Sair

Insira o voo que pretende reservar: _
```

O utilizador insere a opção do voo (neste exemplo, números de 1 a 3 ou 0). Se o utilizador escolheu um número de entre as opções de voos apresentadas (neste caso, entre 1 e 3), dá **so_success <opção>**. Caso contrário, dá **so_error** e termina.

S2.1.4. O programa pede ao utilizador o seu **<ID_passageiro>**:

```
Insira o ID do seu utilizador: _
```

O utilizador insere o respetivo ID de passageiro (dica: UserId Linux). Se esse ID não estiver registado no ficheiro **passageiros.txt**, dá **so_error** e termina. Caso contrário, reporta **so_success <ID_passageiro>**.

S2.1.5. O programa pede ao utilizador a sua **<Senha>**:

```
Insira a senha do seu utilizador: _
```

O utilizador insere a respetiva senha. Caso o script veja que essa senha não é a registada para esse passageiro no ficheiro **passageiros.txt**, dá **so_error** e termina. Caso contrário, reporta **so_success**.

S2.2. Processamento da resposta:

S2.2.1. Valida se o passageiro possui **<Saldo>**, definido no ficheiro **passageiros.txt**, para comprar o bilhete selecionado no passo **S2.1.3**. Se a compra não é possível por falta de saldo, dá **so_error <preço voo> <Saldo>** e termina. Caso contrário, dá **so_success <preço voo> <Saldo>**.

S2.2.2. Subtrai o valor do **<preço voo>** no **<Saldo>** do passageiro, e atualiza o ficheiro **passageiros.txt**. Em caso de erro (e.g., na escrita do ficheiro), dá **so_error** e termina. Senão, dá **so_success <Saldo Atual>**.

S2.2.3. Decrementa uma unidade aos lugares disponíveis do voo escolhidos no passo **S2.1.3**, e atualiza o ficheiro **voos.txt**. Em caso de erro (por exemplo, na escrita do ficheiro), dá **so_error** e termina. Senão, dá **so_success**.

S2.2.4. Regista a compra no ficheiro **relatorio_reservas.txt**, inserido uma nova linha no final deste ficheiro. Em caso de erro (por exemplo, na escrita do ficheiro), dá **so_error** e termina. Caso contrário, dá **so_success**.

O ficheiro **relatorio_reservas.txt** tem o seguinte formato:

<ID_reserva>:<NrVoo>:<Origem>:<Destino>:<Preço>:<ID_passageiro>:<Data Reserva>:<Hora Reserva>

Um exemplo do ficheiro **relatorio_reservas.txt** pode ser:

relatorio_reservas.txt

```
12087:FR5102:Dubai:Bangkok:450:a123456:2024-02-15:12h30
12088:PT5030:Lisboa:Terceira:340:a165432:2024-02-18:09h12
12089:PT6202:Tóquio:Singapura:1700:a124689:2024-02-19:22h37
12090:FR5102:Dubai:Bangkok:450:a165432:2024-02-25:08h48
```

Sendo:

<ID_reserva>: Número inteiro sequencial, deverá ser sempre superior ao da última reserva do ficheiro.

<Data Reserva>: Data em que foi realizada a reserva (compra de bilhete), no formato AAAA-MM-DD.

<Hora Reserva>: Hora em que foi realizada a reserva (compra de bilhete), no formato HHhMM.

S3. Script: **estado_voos.sh**

Este script não recebe nenhum argumento, e é responsável pelo relatório do estado dos voos que pertencem à plataforma **IscteFlight**.

S3.1. Validações:

S3.1.1. O script valida se o ficheiro **voos.txt** existe. Se não existir, dá **so_error** e termina. Senão, dá **so_success**.

S3.1.2. O script valida se os formatos de todos os campos de cada linha do ficheiro **voos.txt** correspondem à especificação indicada em **S2**, nomeadamente se respeitam os formatos de data e de hora. Se alguma linha não respeitar, dá **so_error <conteúdo da linha>** e termina. Caso contrário, dá **so_success**.

S3.2. Processamento do script:

S3.2.1. O script cria uma página em formato HTML, chamada **voos_disponiveis.html**, onde lista os voos com lugares disponíveis, indicando nº, origem, destino, data, hora, o número de lugares disponíveis em cada classe, e o total de lugares disponíveis (para isso deve calcular a soma de todos os lugares disponíveis). Em caso de erro (por exemplo, se não conseguir escrever no ficheiro), dá **so_error** e termina. Caso contrário, dá **so_success**.

O ficheiro **voos_disponiveis.html** tem o seguinte formato:

```
<html><head><meta charset="UTF-8"><title>IscteFlight: Lista de Voos Disponíveis</title></head>
<body><h1>Lista atualizada em <Data Atual, formato AAAA-MM-DD> <Hora Atual, formato HH:MM:SS></h1>
<h2>Voo: <NrVoo>, De: <Origem> Para: <Destino>, Partida em <DataPartida> <HoraPartida></h2>
<ul>
<li><b>Lotação:</b> <Lotação> Lugares</li>
<li><b>Lugares Disponíveis:</b> <Lugares Disponíveis> Lugares</li>
<li><b>Lugares Ocupados:</b> <Lugares Ocupados> Lugares</li>
</ul>
<h2>Voo: FR2101, De: Lisboa Para: Dubai, Partida em 2024-05-23 08h00</h2>
<ul>
<li><b>Lotação:</b> 210 Lugares</li>
<li><b>Lugares Disponíveis:</b> 35 Lugares</li>
<li><b>Lugares Ocupados:</b> 175 Lugares</li>
</ul>
<h2>Voo: <NrVoo>, De: <Origem> Para: <Destino>, Partida em <DataPartida> <HoraPartida></h2>
...
</ul>
</body></html>
```

O ficheiro **voos_disponiveis.html** foi disponibilizado na diretoria de trabalho, como exemplo para perceberem o formato do mesmo. Para testar o mesmo, e assumindo que a diretoria de trabalho foi definida como estando em `~/parte-1/` (se não for o caso, altere o texto seguinte para ficar coerente), escreva a seguinte sequência de comandos (é necessário apenas fazer isto uma vez em todo o trabalho):

```
$  mkdir ~/public_html && cd ~/public_html && ln -s ~/parte-1/voos_disponiveis.html
```

Estes comandos criam a vossa diretoria default web-server (`public_html`, o nome tem de ser esse!), entra nessa diretoria, e cria um soft-link nessa diretoria para o ficheiro `voos_disponiveis.html` que deve estar na sua área de trabalho. Se a sua área de trabalho não for `~/parte-1/`, então terá de mudar o comando de forma correspondente.

Verifique que o ficheiro foi bem produzido, colocando o seguinte endereço no seu web browser (Google Chrome, Firefox ou outro), substituindo **axxxxx** pelo seu UserId Linux no Tigre:

`http://tigre.iul.lab/~axxxxx/voos_disponiveis.html`

S3.3. Invocação do script `estado_voos.sh`:

S3.3.1. Altere o ficheiro **`cron.def`** fornecido, por forma a configurar o seu sistema para que o **script** seja executado de hora em hora, diariamente. Nos comentários no início do ficheiro **`cron.def`**, explique a configuração realizada, e indique qual o comando que deveria utilizar para despoletar essa configuração. O ficheiro **`cron.def`** alterado deverá ser submetido para avaliação juntamente com os outros Shell scripts.

S4. Script: **stats.sh**

Este script obtém informações sobre o sistema, afixando resultados diferentes no STDOUT consoante os argumentos passados na sua invocação. A sintaxe resumida é: **./stats.sh <passageiros>|<top <nr>>**

S4.1. Validações:

S4.1.1. Valida os argumentos recebidos e, conforme os mesmos, o número e tipo de argumentos recebidos. Se não respeitarem a especificação, dá **so_error** e termina. Caso contrário, dá **so_success**.

S4.2. Invocação do script:

S4.2.1. Se receber o argumento **passageiros**, (i.e., **./stats.sh passageiros**) cria um ficheiro **stats.txt** onde lista o nome de todos os utilizadores que fizeram reservas, por ordem decrescente de número de reservas efetuadas, e mostrando o seu valor total de compras. Em caso de erro (por exemplo, se não conseguir ler algum ficheiro necessário), dá **so_error** e termina. Caso contrário, dá **so_success** e cria o ficheiro. Em caso de empate no número de reservas, lista o primeiro do ficheiro. Preste atenção ao tratamento do singular e plural quando se escreve "reserva" no ficheiro). Um exemplo do ficheiro **stats.txt** será:

stats.txt

```
Nuno Garrido: 5 reservas; 5635€
David Gabriel: 2 reservas; 1322€
Fábio Cardoso: 1 reserva; 1720€
```

S4.2.2. Se receber o argumento **top <nr:number>**, (e.g., **./stats.sh top 4**), cria um ficheiro **stats.txt** onde lista os **<nr>** (no exemplo, os 4) voos mais rentáveis (que tiveram melhores receitas de vendas), por ordem decrescente. Em caso de erro (por exemplo, se não conseguir ler algum ficheiro necessário), dá **so_error** e termina. Caso contrário, dá **so_success** e cria o ficheiro. Em caso de empate, lista o primeiro do ficheiro; o ficheiro **stats.txt** ficará então:

stats.txt

```
FR5102: 125779€
PT5000: 98354€
FR2301: 23492€
PR3201: 12145€
```

S5. Script: **menu.sh**

Este script invoca os scripts restantes, não recebendo argumentos. **Atenção:** Não é suposto que volte a fazer nenhuma das funcionalidades dos scripts anteriores. O propósito aqui é simplesmente termos uma forma centralizada de invocar os restantes scripts.

S5.1. Apresentação:

S5.1.1. O script apresenta (pode usar **echo**, **cat** ou outro, sem "limpar" o ecrã) um menu com as opções abaixo indicadas.

```
MENU:
1: Regista/Atualiza saldo do passageiro
2: Reserva/Compra de bilhetes
3: Atualiza Estado dos voos
4: Estatísticas – Passageiros
5: Estatísticas – Top Voos + Rentáveis
0: Sair

Opção: _
```

S5.2. Validações:

S5.2.1. Aceita como input do utilizador um número. Valida que a opção introduzida corresponde a uma opção válida. Se não for, dá **so_error <opção>** (com a opção errada escolhida), e volta ao passo **S5.1** (ou seja, mostra novamente o menu). Caso contrário, dá **so_success <opção>**.

S5.2.2. Analisa a opção escolhida, e mediante cada uma delas, deverá invocar o sub-script correspondente descrito nos pontos **S1** a **S4** acima. No caso das opções **1** e **4**, este script deverá pedir interactivamente ao utilizador as informações necessárias para execução do sub-script correspondente, injetando as mesmas como argumentos desse sub-script:

S5.2.2.1. Assim sendo, no caso da opção **1**, o script deverá pedir ao utilizador sucessivamente e interactivamente os dados a inserir:

```
MENU:
1: Regista/Atualiza saldo do passageiro
2: Reserva/Compra de bilhetes
3: Atualiza Estado dos voos
4: Estatísticas – Passageiros
5: Estatísticas – Top Voos + Rentáveis
0: Sair

Opção: 1

Regista passageiro / Atualiza saldo passageiro:
Indique o nome: Fábio Cardoso
Indique a senha: 12qwaszx
Para registar o passageiro, insira o NIF:
Indique o saldo a adicionar ao passageiro: 500
_
```

Repare que, no exemplo dado acima, não foi inserido qualquer NIF, portanto, indicando que não pretende registar o passageiro, mas sim apenas adicionar saldo ao mesmo, correspondendo esta introdução assim ao primeiro exemplo de invocação via argumentos indicada em **S1**, que se destina apenas a adicionar 500 euros ao saldo do passageiro já existente Fábio Cardoso, validando essa adição de créditos pela inserção da senha do passageiro. Este script não deverá fazer qualquer validação dos dados inseridos, já que essa validação é feita no script **S1**. Após receber os dados, este script invoca o Sub-Script: **`regista_passageiro.sh`** com os argumentos recolhidos do utilizador. Após a execução do sub-script, dá **`so_success`** e volta ao passo **S5.1**.

S5.2.2.2. No caso da opção **2**, o script invoca o Sub-Script: **`compra_bilhete.sh`**. Após a execução do sub-script, dá **`so_success`** e volta para o passo **S5.1**.

S5.2.2.3. No caso da opção **3**, o script invoca o Sub-Script: **`estado_voos.sh`**. Após a execução do sub-script, dá **`so_success`** e volta para o passo **S5.1**.

S5.2.2.4. No caso da opção **4**, o script invoca o Sub-Script: **`stats.sh`** com o argumento necessário. Após a execução do sub-script, dá **`so_success`** e volta para o passo **S5.1**.

S5.2.2.5. No caso da opção **5**, o script deverá pedir ao utilizador o nº de voos a listar, antes de invocar o Sub-Script: **`stats.sh`**:

```
MENU:
1: Regista/Atualiza saldo do passageiro
2: Reserva/Compra de bilhetes
3: Atualiza Estado dos voos
4: Estatísticas – Passageiros
5: Estatísticas – Top Voos + Rentáveis
0: Sair

Opção: 5

Estatísticas – Top Voos + Rentáveis:
Indique o número de voos a listar: _
```

Após a execução do Sub-Script: **`stats.sh`**, dá **`so_success`** e volta para o passo **S5.1**.

Apenas a opção **`0`** (zero) permite sair deste Script: **`menu.sh`**. Até escolher esta opção, o menu deverá ficar em ciclo, permitindo realizar múltiplas operações iterativamente (e não recursivamente, ou seja, não deverá chamar o Script: **`menu.sh`** novamente).

Anexo A: Macros de suporte ao trabalho

Macros fornecidas, no ficheiro **so_utils.sh**, com Mensagens de **sucesso**, **erro, debug** e **validação de Scripts**:

Mensagens de output com Erro (com exemplos): Macro `so_error <passo> <argumentos>`

- Exemplo de **S2.1.1**: **`so_error`**

  `so_error S2.1.1`
- Exemplo de **S3.1.2**: **`so_error <conteúdo da linha>`**

  `so_error S3.1.2 "FR5102:Dubai:Bangkok:11/03/2024:12h30:450:20:30:170"`
- Exemplo de **S2.2.1**: **`so_error <preço voo> <Saldo>`**

  `so_error S2.2.1 1350 1200`

Mensagens de output com Sucesso (com exemplos): Macro `so_success <passo> <argumentos>`

- Exemplo de **S2.1.5**: **`so_success`**

  `so_success S2.1.5`
- Exemplo de **S2.1.2**: **`so_success <Cidade>`**

  `so_success S2.1.2 Bangkok`
- Exemplo de **S2.2.1**: **`so_success <preço voo> <Saldo>`**

  `so_success S2.2.1 1350 5200`

Mensagens de Debug (com exemplos): Apesar de não ser necessário, disponibilizou-se também uma macro para as mensagens de debug dos scripts, dado que será muito útil aos alunos: **Macro `so_debug <argumentos>`**

Um exemplo de utilização nos scripts é, **`var1=3; so_debug "var1:$var1"`** # mostra "@DEBUG {…} [var1:3]"

Tem a vantagem de que mostra sempre as mensagens de debug (não precisa sequer ser nunca apagado). Quando os alunos quiserem que essas mensagens de debug não apareçam, simplesmente tiram o comentário da primeira linha do Shell script onde estão a trabalhar:

```
# export SO_HIDE_DEBUG=1            ## Uncomment this line to hide all @DEBUG statements
```

e podem verificar que, mesmo que as invocações à macro **so_debug** estejam no vosso código, nenhuma mensagem de @DEBUG será mostrada, e, assim, não precisam de apagar as invocações à macro **so_debug**, mantendo os vossos scripts intocados. A macro **so_debug** tem ainda a enorme vantagem de mostrar o script e a linha de código que invocou esta macro, assim podem ter a certeza de saber sempre onde estão no vosso script.

Para terem a completa experiência destas (e outras macros utilitárias), sugere-se que experimentem o script Shell disponível no Tigre, executando:

```
$ /home/so/utils/bin/so_utils_tester.sh
```

Script Validador do trabalho:

Como anunciado nas aulas, está disponível para os alunos um script de validação dos trabalhos, para os alunos terem uma noção dos critérios de avaliação utilizados.

Passos para realizar a validação do seu trabalho:

- Garanta que o seu trabalho (i.e., os cinco scripts **`.sh`**) está localizado numa diretoria local da sua área. Para os efeitos de exemplo para esta demonstração, assumiremos que essa diretoria terá o nome **`parte-1`** (mas poderá ser outra qualquer).
- Posicione-se nessa diretoria **`parte-1`** da sua área:

  ```
  $ cd parte-1
  ```

- Dê o comando seguinte e valide que está mesmo na diretoria correta:

  ```
  $ pwd
  ```

- Dê o comando seguinte, e confirme que todos os ficheiros **`.sh`** do seu trabalho estão mesmo nessa diretoria, que esses ficheiros têm permissão de execução, e que nessa diretoria existe uma subdiretoria chamada **`so_2023_trab1_validator`**:

  ```
  $ ls -l
  ```

- Vá para a subdiretoria onde está o validador:

  ```
  $ cd so_2023_trab1_validator
  ```

- E, finalmente, dentro dessa diretoria, executem o script de validação do vosso trabalho:

  ```
  $ ./so_2023_trab1_validator.py ..
  ```

- Resta agora verificar quais dos seus testes "passam" (✓) e quais "chumbam" (✗).
- Faça as alterações para correção dos seus scripts.
- Sempre que quiser voltar a fazer nova validação, basta novamente posicionar-se na subdiretoria **`so_2023_trab1_validator`** e correr o script de validação como demonstrado acima.

Anexo B: Histórico de Versões

Versão 1: publicada em 2024-02-25

- Primeira versão (original) do trabalho;

Versão 2: publicada em 2024-02-26

- Retiradas as classes dos voos, passa a haver apenas uma classe única;
- O ficheiro estado_voos.txt passou a ser obsoleto, vai ser removido, a informação de disponibilidade passa a constar do ficheiro voos.txt;
- A informação da classe no relatório de reservas foi substituída pelo preço do voo;
- Cron Job passou a ser feito diariamente.