Estruturas de Dados e Algoritmos

Projeto prático 2 - 2023/2024

(15% da avaliação da UC)

Código de ética e honestidade académica

Nesta disciplina, espera-se que cada aluno subscreva os mais altos padrões de honestidade académica. Isto significa que cada ideia que não seja do aluno deve ser explicitamente creditada ao(s) respetivo(s) autor(es). O não cumprimento do disposto constitui uma prática de plágio. O plágio inclui a utilização de ideias, código ou conjuntos de soluções de outros alunos ou indivíduos, ou qualquer outra fonte para além dos textos de apoio à disciplina, sem dar o respetivo crédito a essas fontes. A menção das fontes não altera a classificação, mas os alunos não deverão copiar código de outros colegas, ou dar o seu próprio código a outros colegas em qualquer circunstância. De notar que a responsabilidade de manter o acesso ao código somente para os colegas de grupo é de todos os elementos.

É crucial que todos os trabalhos submetidos neste curso sejam o resultado do vosso próprio esforço e entendimento. O uso do ChatGPT ou de qualquer outra ferramenta de assistência é permitido, desde que seja para esclarecer conceitos e apoiar a aprendizagem, não para obter soluções prontas.

Fica explicitamente proibido submeter qualquer trabalho que tenha sido gerado pelo ChatGPT ou outros modelos de geração de texto, sem que o aluno seja capaz de explicar de forma clara e precisa o código produzido. A não observância deste aviso será considerada uma violação grave do código de ética académica e resultará na reprovação do projeto prático e disciplina.

Reforçamos que o objetivo deste curso não é apenas obter resultados, mas sim desenvolver as vossas habilidades individuais de programação e raciocínio lógico. O uso indevido de recursos externos compromete não apenas a integridade do processo de aprendizagem, mas também a reputação e credibilidade de cada aluno.

Sejam éticos, sejam responsáveis e demonstrem o vosso compromisso com a aprendizagem genuína. Para qualquer dúvida ou questão a equipa docente está à vossa disposição para ajudar.

1. Objetivos

O objetivo do projeto é o desenvolvimento de um programa em C++ que simule o funcionamento do "**aeroporto EDA**". O sistema deverá implementar todas as atividades básicas de um aeroporto para que a simulação possa ser usada por entidades externas como um modelo de funcionamento de qualquer aeroporto no mundo. De uma forma geral o programa deve suportar:

- Representação dos aviões em aproximação
- Representação dos aviões em pista
- Representação dos aviões a descolar
- Representação dos passageiros
- Identificação/pesquisa dos aviões por voo ou modelo
- Identificação/pesquisa dos passageiros por nome ou número de bilhete
- Atividades de emergência
    - Alteração das prioridades de aterragem e descolagem
- **A utilização do paradigma orientado a objetos (classes) não é permitida**

Espera-se que a funcionalidade descrita neste enunciado seja implementada com a ajuda de: **Listas Ligadas** (listas de voo, em aproximação, em pista, e a descolar), **Árvores de Pesquisa Binária** (passageiros em pista) e **Arrays Dinâmicos** (conteúdo dos ficheiros), entre outras estratégias de implementação discutidas nas aulas.

O esquema abaixo exemplifica como é que estas estruturas deverão ser organizadas para a implementação do projeto:

- Listas ligadas para os 3 conjuntos de aviões que deverão ser registados (chegada, em pista e partidas).
- Listas Ligadas para guardar os passageiros associados a cada avião.
- Lista de chegada de nacionalidades, em que cada nó contém a raiz de uma árvore de pesquisa binária de Passageiros (a estrutura Passageiro utilizada aqui poderá não ser a mesma da utilizada em cada avião).

2. Descrição

As principais entidades do programa são o avião e o passageiro, abaixo é apresentada a sua descrição.

Cada **avião** é representado por:

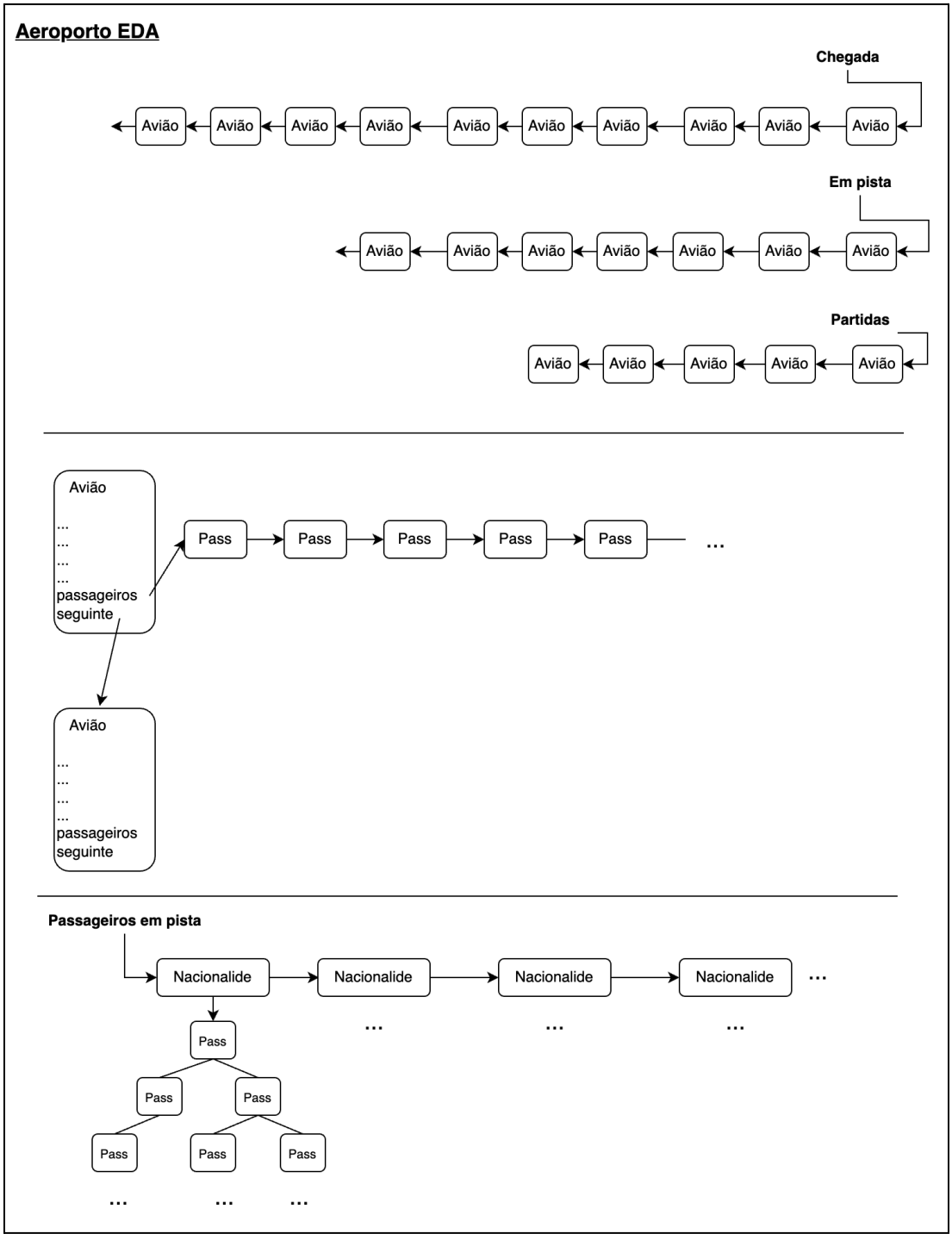
- Nome do voo
- Modelo do avião
- Origem
- Destino
- Capacidade
- Conjunto de passageiros

(de forma a simplificar a aplicação vamos assumir que os voos estão sempre cheios e que a capacidade varia entre 5 e 15 passageiros)

Cada **passageiro** é identificado por:

- Número do bilhete
- Primeiro nome
- Segundo nome
- Nacionalidade

NOTA: A descrição acima não se refere a nenhuma estrutura de dados (struct) em C++, cabe a cada grupo escolher a melhor implementação para as entidades avião e passageiro.


Aeroporto EDA
Chegada
Avião
Em pista
Avião
Partidas
Avião
Avião
...
...
...
...
passageiros
seguinte
Pass
Avião
Passageiros em pista
Nacionalide
Pass

Esquema 1

3. Funcionamento

3.1 Funcionamento geral

A simulação do aeroporto desenvolve-se por ciclos desencadeados pelo utilizador. Cada vez que o utilizador pressionar a tecla "**s + enter**" um novo ciclo começa. Em cada ciclo o sistema deve (por esta ordem):

1. Remover um avião ao conjunto dos aviões a descolar.
2. Mover um avião do conjunto de aviões em pista para o conjunto de aviões a descolar
3. Mover um avião do conjunto de aviões em aproximação para o conjunto de aviões em pista
4. Adicionar um avião ao conjunto dos aviões em aproximação.

Em situações normais o avião que está há mais tempo em aproximação é próximo a aterrar (passa para o conjunto de aviões em pista), da mesma forma o avião que está há mais tempo na pista é o próximo a descolar (passa para o conjunto dos aviões a descolar). O avião que descolou há mais tempo é removido do sistema a cada ciclo.

O sistema mantém registo dos 10 próximos aviões a aterrar, o aeroporto tem capacidade para 7 aviões em pista, e são mantidos registos dos últimos 5 aviões a descolar.

NOTA: De forma a manter a organização em pista, o aeroporto requer que os aviões em pista estejam ordenados pelo número de voo.

3.1.1 Inicialização

De forma a facilitar o funcionamento do programa, assumimos que:

- Antes de qualquer avião aterrar o conjunto de aviões em aproximação tem de estar completo (10 aviões)
- Antes de qualquer avião descolar o conjunto de aviões em pista tem de estar completo (7 aviões)

3.1.2 Persistência dos dados

A qualquer momento entre os ciclos o operador do programa pode pressionar a tecla **'g'** e gravar todos os dados do aeroporto. Cada grupo é livre de implementar o seu método de gravação. No entanto, o sistema deverá permitir o carregamento dos dados.

3.1.3 Persistência dos dados - linha de comandos

Deverá também ser possível carregar o aeroporto, caso o nome dos ficheiros com os dados do aeroporto gravado, sejam passados pela linha de comandos quando o programa for inicializado.

3.2 Criação de aviões e passageiros

A geração de aviões e passageiros deverá ser feita **de forma aleatória** a cada ciclo.

A geração de aviões é feita apenas quando estes são adicionados ao conjunto de aviões em aproximação, para auxiliar neste processo o sistema deverá utilizar 5 ficheiros externos *modelos.txt*, *voo.txt, origem.txt, destino.txt*, que contêm modelos, voos, origens, destinos separados por **"enters".** O sistema deverá selecionar os dados dos ficheiros aleatoriamente e criar um avião. A capacidade de um avião deverá ser um valor aleatório entre 10 a 15 passageiros.

NOTA: Todos os aviões em aproximação têm como destino o "aeroporto EDA". Esse destino é alterado quando eles passam para o conjunto de aviões a descolar, e nesse caso o nome do destino deve ser retirado aleatoriamente do ficheiro destinos.txt.

Cada avião possui **também** um conjunto de passageiros associados. A criação dos passageiros é feita de forma idêntica aos aviões. Os ficheiros *primeiro_nome.txt, segundo_nome.txt e nacionalidade.txt* serão usados para auxiliar a criação de passageiros, ou seja, o primeiro, último nome e nacionalidade dos passageiros será um valor aleatório retirado dos ficheiros indicados acima. O número do bilhete deverá ser um texto do tipo "TKXXXXXXXXXX" em que os X poderão ser qualquer número inteiro.

Quando um avião chega à pista o seu número de voo também é alterado (deverá ser selecionado outro voo do ficheiro voo.txt)

3.2.1 Ficheiros

Os ficheiros *primeiro_nome.txt, segundo_nome.txt, nacionalidade.txt, modelos.txt*, *voo.txt, origem.txt, destino.txt* são iguais para todos os grupos e estão disponíveis aqui.

3.3 Situações de emergência

Em situações de emergência o funcionamento sequencial do aeroporto é alterado. Neste caso, o operador do programa terá que identificar qual o voo em emergência e terá também que assegurar que existe espaço em pista para esse avião. Estas operações acontecem entre os ciclos de execução. Para isto, o operador deverá pressionar a tecla '**e**'. O sistema deverá pedir o número do voo que precisa de aterrar de emergência. De seguida, o sistema deverá pedir o número do voo que terá que descolar de forma a haver espaço para o avião em emergência aterrar. Após essa alteração o sistema deverá:

1. Gerar um novo voo e adicionar ao final do conjunto de aviões em aproximação (de forma a completar este conjunto)
2. Voltar para o estado anterior (representado no esquema 1), mas já com os voos atualizados.

3.4 Movimentação de passageiros

A cada chegada de um avião os seus passageiros são removidos do avião(chegaram ao destino).

No entanto, enquanto este avião se encontra em pista um novo conjunto de passageiros é criado e associado ao avião.

3.5 Passageiros no aeroporto

Os passageiros de nacionalidade estrangeira que cheguem ao "**aeroporto EDA**" devem ser mantidos numa lista ligada em que cada nó da lista identifica uma nacionalidade. Cada nó é também raiz de uma árvore de pesquisa binária onde deverão ser inseridos os passageiros organizados pelo seu nome.

3.5.1 Aquando da criação da lista de passageiros para uma nova partida (ver ponto 3.4) deverão ser adicionados 3 passageiros estrangeiros que estão neste conjunto. A remoção destes passageiros poderá ser implementada através da remoção por cópia (ver ficheiro aqui).

3.6 Fechar o aeroporto

Deverá ser possível ao operador do programa fechar o aeroporto por um número de ciclos que nunca deverá ser superior a 5. Nestes casos os novos voos que estão a ser gerados deverão ficar em espera nas chegadas (nesta situação em que o aeroporto está fechado a dimensão da lista de chegada poderá, excepcionalmente, ultrapassar os 10 aviões). Após o aeroporto voltar ao funcionamento normal, e de forma a *despachar* os próximos ciclos deverão tratar de 2 aviões simultaneamente (adicionar e remover da pista) até a lista de chegada voltar ao seu tamanho normal.

3.7 Inverter Prioridade

Deverá ser possível ao operador inverter a prioridade dos aviões em pista. Na prática após esta operação o avião que estaria na última posição da fila, e por consequência seria o último a descolar, deverá passar para início da mesma, e será o próximo a descolar (a mesma lógica deverá ser repetida para todos os aviões em pista).

4. Visualização

A visualização do aeroporto na consola deverá seguir o seguinte formato (repetido a cada ciclo)

```
(e)mergências (o)pções (g)ravar
---------------
Em aproximação
---------------
Voo TP1193
Modelo: A320
Origem:Lisboa
Destino: Aeroporto EDA
Passageiros: Quintal, Alves, Gouveia, Pereira, Esteves,...
.
.
.
--------------
Na pista
--------------
Voo TP1113
Modelo: B767
Origem:Aeroporto EDA
Destino: Brasil
Passageiros: Gaucho, Jesus, Alves, Scolari,...
.
.
.
-------------
A descolar
-------------
Voo TP1141
Modelo: B
Origem:Aeroporto EDA
Destino: Paris
Passageiros: Martin, Bernard, Dubois, Simon, Bertrand,...
```

Figura 1 : Exemplo de visualização

5. Outras funcionalidades

Entre cada ciclo o sistema deverá permitir as seguintes operações, estas deverão ser apresentadas num menu caso o operador pressione a tecla '**o**'

5.1 Mostrar os passageiros em pista

5.2 Mostrar todos os passageiros em pista por:

5.2.1 Nacionalidade, ordenados por ordem alfabética

5.2.2 Nacionalidade, utilizando uma representação visual para uma árvore de pesquisa binária (ver ficha prática 13)

5.3 Pesquisa sobre os passageiros nas chegadas e partidas (pelo primeiro nome)

5.4 Editar a nacionalidade de passageiro num voo das chegadas

6. Documentação do trabalho

Todas as funções terão de estar devidamente documentadas de acordo com o *template* automático gerado pelo Clion. Nesta documentação deverão estar claramente identificados os argumentos, e tipo de retorno da função. Bem como uma breve descrição do funcionamento da função:

```
/**
* Função que recebe um array de inteiros e retorna a soma dos elementos do
array recebidos
* @param v - array cujo os valores serão somados
* @param tamanho - tamanho do array recebido
* @return - soma do valores de v
*/
float somarElementos(int v[], int tamanho){
   float soma=0;
   for(int i=0;i<tamanho;i++){
       soma = soma+v[i];
   }
   return soma;
}
```

Além da documentação do código, juntamente com o projeto deverá ser entregue um documento que indica a divisão das tarefas de implementação pelos membros do grupo

a. A escrita da documentação não é considerada uma tarefa de implementação
b. Membros de grupo sem tarefas atribuídas/realizadas reprovam o projeto prático da disciplina

7. Entrega

As entregas do projeto serão finalizadas no moodle, na data da entrega será criado um formulário no moodle em que cada grupo deverá submeter um ficheiro .zip ou .rar com o relatório e todo o código e ficheiros necessários para a execução/avaliação.
O nome do ficheiro entregue deverá ser o mesmo do número do grupo, por exemplo Grupo1.zip. Para os alunos que desenvolveram o projeto no Visual Studio, terão que eliminar a pasta oculta .vs na raiz do projeto antes de comprimir o projeto em .zip ou .rar.

7.1 Datas Entregas

1ª Entrega: A primeira entrega está agendada para o dia 30 de Abril de 2024, nesta entrega deverão ser entregues as seguintes funcionalidades:

- 2, 3.1 e 3.2 e todos os menus do programa.
- A primeira entrega é **obrigatória** e poderá influenciar até 15% da avaliação final
  - A não implementação de cada requisito irá prejudicar a avaliação **final** em 5%, até um total de 15%
- Nesta entrega espera-se uma implementação mínima mas funcional dos requisitos acima. Nesta fase é omitido todo o movimento de passageiros, ou seja, os passageiros gerados na criação do voo metêm-se no mesmo até o voo ser removido do sistema (sair das partidas).

2ª Entrega: A segunda entrega está agendada para dia 25 de Maio de 2024, e deverão ser entregues todas as funcionalidades.

- Nesta entrega, se entenderem, os grupos poderão implementar alterações às funcionalidades entregues na primeira entrega.

Todas as entregas do projeto entregue deverão compilar sem problemas, projetos que não executem serão ignorados. É preferível entregar um projeto mais simples mas funcional, do que um projeto complexo que não seja possível testar

7.2 Defesas dos projetos

Na semana da segunda entrega serão agendadas reuniões com os membros de cada grupo de forma a aferir a contribuição de cada membro para o projeto. Esta reunião pode influenciar a nota individual até 100%, na prática isto significa que no mesmo grupo poderão haver colegas avaliados em 18 enquanto que outros poderão ter uma avaliação abaixo da nota mínima (**o que implica a reprovação da disciplina**). De forma a facilitar esta avaliação individual, é **obrigatório** que todos os grupos mantenham o registo de **todas** as tarefas desenvolvidas no âmbito do projeto (ver ponto 6 do relatório).

7.3 Critérios de avaliação

- Definição de estruturas de dados adequadas;
- Cumprimento dos objetivos;
- Qualidade do código desenvolvido;
- Qualidade de execução do programa desenvolvido;
- Relatório e documentação do código;
- Apresentações/discussões.

8. Notas implementação

Durante a inicialização do programa existem várias variáveis (por exemplo quantidade de secções, e capacidade das mesmas) que serão aleatórias.
Para implementar este comportamento espera-se a utilização da função **`srand`**, a utilização deverá seguir os seguintes passos:

- **No "main" do programa** : importar as bibliotecas **`<stdlib.h>`** e **`<time.h>`**
  - criar o "seed" para a função rand através da instrução **`srand (time(NULL));`**
- Para o cálculo de um número aleatório (em qualquer ficheiro)
  - importar o **`<stdlib.h>`**
  - Utilizar a função **`rand()%NÚMERO`** para calcular um valor aleatório entre 0 e um (**NÚMERO -1**) definido

```
/* valor aleatório entre 1 e 10: */
  valor = rand() % 10 + 1;
```

Cada grupo é livre de implementar soluções para todos os restantes detalhes de implementação, não definidos neste enunciado.