# Instituto Politécnico do Cávado e Ave



# Trabalho Prático 2

# Relatório

# Jorge Miguel Arezes Noro | 15705

# Engenharia e Desenvolvimento de Jogos Digitais

# Paradigmas de Programação II

# Docente: Luís Gonzaga Martins Ferreira

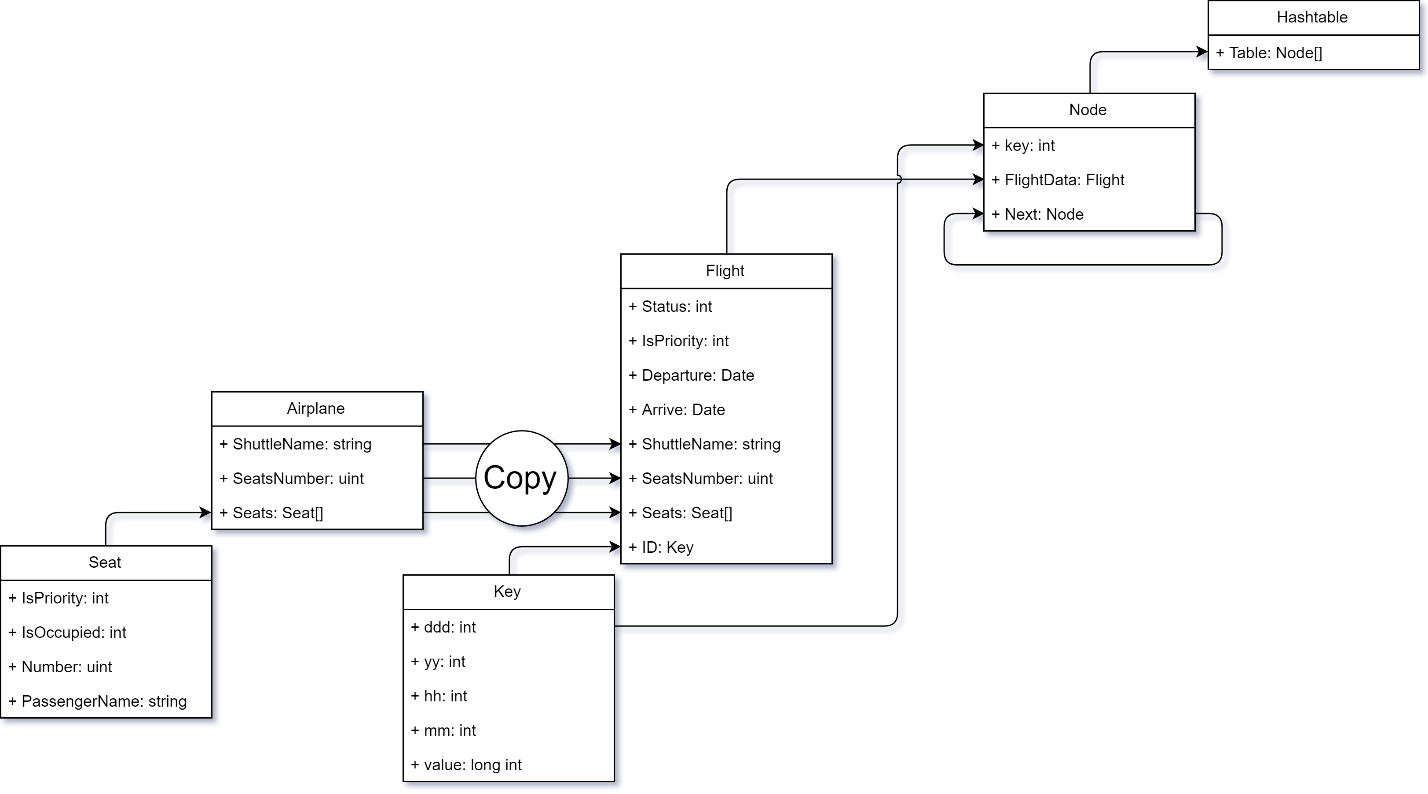
2º Semestre

Introdução

Este documento serve de relatório ao Trabalho Prático 2 da unidade curricular de Paradigmas de Programação II. É acompanhado de um projeto de Visual Studio (*pp2-tp2.sln*). Não existem dependências a projetos nem bibliotecas externas.

O projeto está disponível no GitHub, no seguinte URL:   
<https://github.com/mrMav/pp2-tp2>

1. Alocação de memória

Abaixo fica represado em *pseudo* UML a relação das estruturas de dados:  


As estruturas de dados para armazenar os dados são dinâmicas, sendo constituídas por listas ligadas e hashtables. A hashtable é do tipo *open adressing*, em que cada bloco do *array* consiste de uma lista.

A tabela tem um tamanho de 367 blocos. Este tamanho tem como base o número de dias de ano bissexto (arredondado para o número primo mais próximo). Ao existirem tantos blocos como dias de ano, é nos permitido então usar o número do dia do ano como chave. (por exemplo, 01/01/2018 é o dia 1 e 01/06/2018 é o 149).

Com isto, obtemos uma *hastable* em que as “colunas” correspondem a anos completos.

Cada voo possui também um ID, que é útil para o utilizador selecionar o voo pretendido. O padrão do ID é “dddaahhmm”. Dia do ano, últimos dois dígitos do ano, horas (formato 24h) e minutos. (como exemplo, o voo do dia 01/06/2018 às 18h20, tem como ID 149181820).

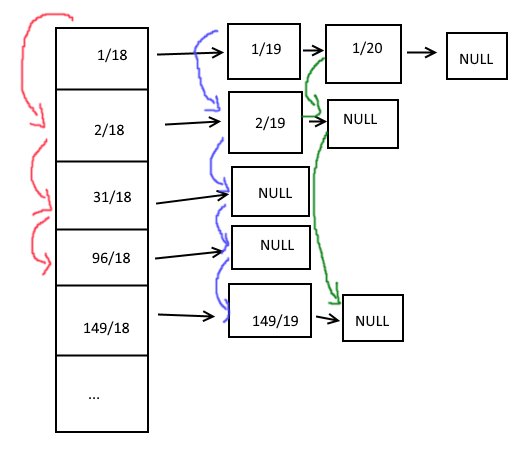
Existe uma “classe” de avião, para ser possível aumentar a frota da companhia e utilizar distintas naves para cada voo.

As funções de alocação de memória podem ser consultadas no ficheiro *CreateFunctions.c*.

1. Algoritmos

No que diz respeito a algoritmos, um dos mais interessantes é implementado na função *Flight\* GetClosestFlightHoldPassengers(HashTable\* ht, int n, Airplane\* plane)*, é possível ver a usa implementação completa no ficheiro *UserFunctions.c*.

Como a *hashtable* está organizada em colunas por ano, quando queremos atravessar a tabela por ordem cronológica (sendo que temos um voo por dia), temos de o fazer uma coluna de cada vez. A imagem abaixo tenta ilustrar isso mesmo:



A ideia será primeiro seguir as setas vermelhas, depois as azuis, as verdes e assim sucessivamente. Com este algoritmo, conseguimos então correr por ordem cronológica todos os voos e realizar operações conforme necessário.

Sempre que um voo é adicionado, a lista é organizada de forma crescente, em que o parâmetro de comparação é a data, convertida em milissegundos (*UNIX Epoch time*). O algoritmo implementado é do tipo *bubble sort* e pode ser analisado na integra no ficheiro *Hashtable.c* na *função Node\* ListSortAscending(Node\* head)*.