



# **Aeroportos Low-Cost O'Connor**

---

Turma 2 – Grupo 7  
Eduardo Brito - up201806271  
Pedro Ferreira - up201806506  
Pedro Ponte - up201809694



# PARTE 1

Estruturas de dados lineares

**Visualizar Parte 2**

## **Solução e algoritmos relevantes**

Descrição da solução encontrada para o gerenciamento da empresa

01

## **Diagrama de Classes**

Diagrama de classes do trabalho desenvolvido

02

## **Estrutura de Ficheiros**

Forma como os dados são guardados em ficheiros

03

# **Índice**

04

## **Tratamento de Exceções**

Apresentação da forma como as exceções foram tratadas

05

## **Funcionalidades Implementadas**

Amostragem de Menus CRUD, listagens e pesquisas

06

## **Observações Finais**

Destaque de uma funcionalidade implementada e Dificuldades encontradas

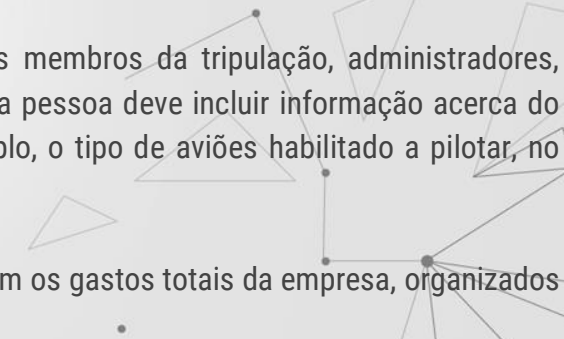


# 01

**Solução e algoritmos  
relevantes**

# Descrição do problema

O problema proposto baseia-se essencialmente na criação de uma aplicação na linguagem de programação C++, para o gerenciamento de uma empresa de aeroportos, onde foi sugerido:

1. A criação de uma rede interligada de aeroportos, da companhia O'Connor, de onde deve ser possível adicionar, remover, consultar e alterar voos, funcionários e aviões.
  2. Para cada aeroporto é necessária a informação do seu gerente, a sua localização, os seus funcionários e aviões.
  3. Para os aviões, há a necessidade de saber a informação sobre a sua categoria, o seu custo de operações, capacidade e os voos previstos.
  4. No que diz respeito aos funcionários, existem distintas categorias, sendo elas membros da tripulação, administradores, pilotos e empregados. Cada uma, para além de apresentar os dados pessoais da pessoa deve incluir informação acerca do salário, e algumas informações específicas de cada categoria, como, por exemplo, o tipo de aviões habilitado a pilotar, no caso dos pilotos.
  5. Os voos devem ser realizados entre aeroportos, com respetiva data e hora.
  6. Por fim, deve ser implementado um menu de consulta das finanças, onde aparecem os gastos totais da empresa, organizados e solicitados por parâmetros.
- 

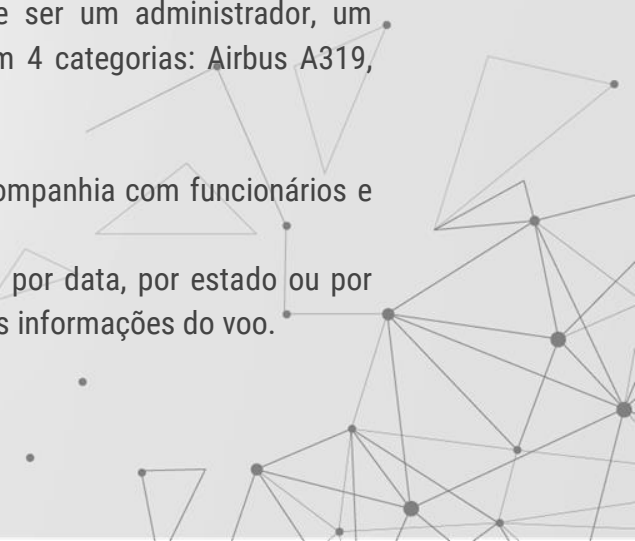
# Resolução do problema

Para solucionar o problema atrás descrito, dividimos este programa em duas partes principais:

- As “CRUD Operations”, onde se pode criar, editar e remover um aeroporto, ou um voo. Podemos ainda ver uma lista de todos os aeroportos e dos seus gestores, ou de todos os voos existentes.
- “Airport Management”, onde se gere cada um dos aeroportos existentes, como adicionar, editar, ou remover um avião, ou um funcionário. Um funcionário pode ser um administrador, um empregado, um assistente de bordo, ou um piloto. Os aviões têm 4 categorias: Airbus A319, Airbus A320, Airbus A330 e Boeing 737.

Foi criada também uma área financeira, onde se mostram os gastos totais da companhia com funcionários e aviões, ou os gastos distribuídos por cada um dos aeroportos.

Existe ainda um painel de voos, onde é possível pesquisar um determinado voo por data, por estado ou por localização. Também é possível visualizar todos os voos existentes e as respetivas informações do voo.



1

```

class Company{
private:
    string name;
    vector<Airport *> airports = vector<Airport *>();
    vector<Flight *> flights = vector<Flight *>();
    string banner = "";

public:
    Company(const string &name);
    ~Company();

    const string &getName() const { return name; }
    void setName(const string &name) { Company::name = name; }

    const string &getBanner() const { return banner; }
    void setBanner(const string &Banner){ banner = Banner; }

    void addAirport();
    void addFlight();

    void updateAirport();
    void updateFlight();

    void getAirports();
    void getFlights();

    void removeAirport();
    void removeFlight();

    void showFinances();
    float ExpensesValue();

```

```

friend void readFiles(Company &company);
friend void showBanner(Company &company, bool onlyBanner);
friend int showMenu(Company &company);
friend int showCRUDMenu(Company &company);
friend int showAPManagement(Company &company);
friend int showAirportFinances(Company &company);
friend int showFlightsPanel(Company &company);
friend void saveFiles(Company &company);

```

## Main Class:

Os conceitos de friend function foram usados extensivamente para permitir uma maior flexibilidade e acessibilidade do código na criação de Menus e outras funcionalidades operacionais.

2

```

class Airport{
    Administration *manager;
    Location *location;
    vector<Person *> people;
    vector<Plane *> planes;

```

## Funcionamento geral:

Toda a implementação da aplicação é gerida através de classes, recorrendo fortemente aos conceitos de herança e polimorfismo para lidar com toda a informação e todas as operações relacionadas com os objetos, da maneira mais eficiente possível.

## Persistência da Informação:

Para que toda a informação gerada durante o funcionamento do programa pudesse ser mantida em execuções consequentes, foram implementados e organizados vários algoritmos de leitura e escrita em ficheiros, que também utilizam todos os conceitos descritos no ponto anterior.

3

```

Person *p = vPeople.at(index);
ssPeople << p->getName() << ";";
<< p->getBirthDay().getYear() << ";";
<< p->getBirthDay().getMonth() << ";";
<< p->getBirthDay().getDay() << ";";
<< p->getSalary() << ";";
<< p->getCategory() << ";";
if(p->getCategory() == "Admin"){
    ssPeople << dynamic_cast<Administration *>(p)->getDepartment();
}
else if(p->getCategory() == "Employee"){
    ssPeople << dynamic_cast<Employee *>(p)->getSchedule();
}
else if(p->getCategory() == "Crew"){

```

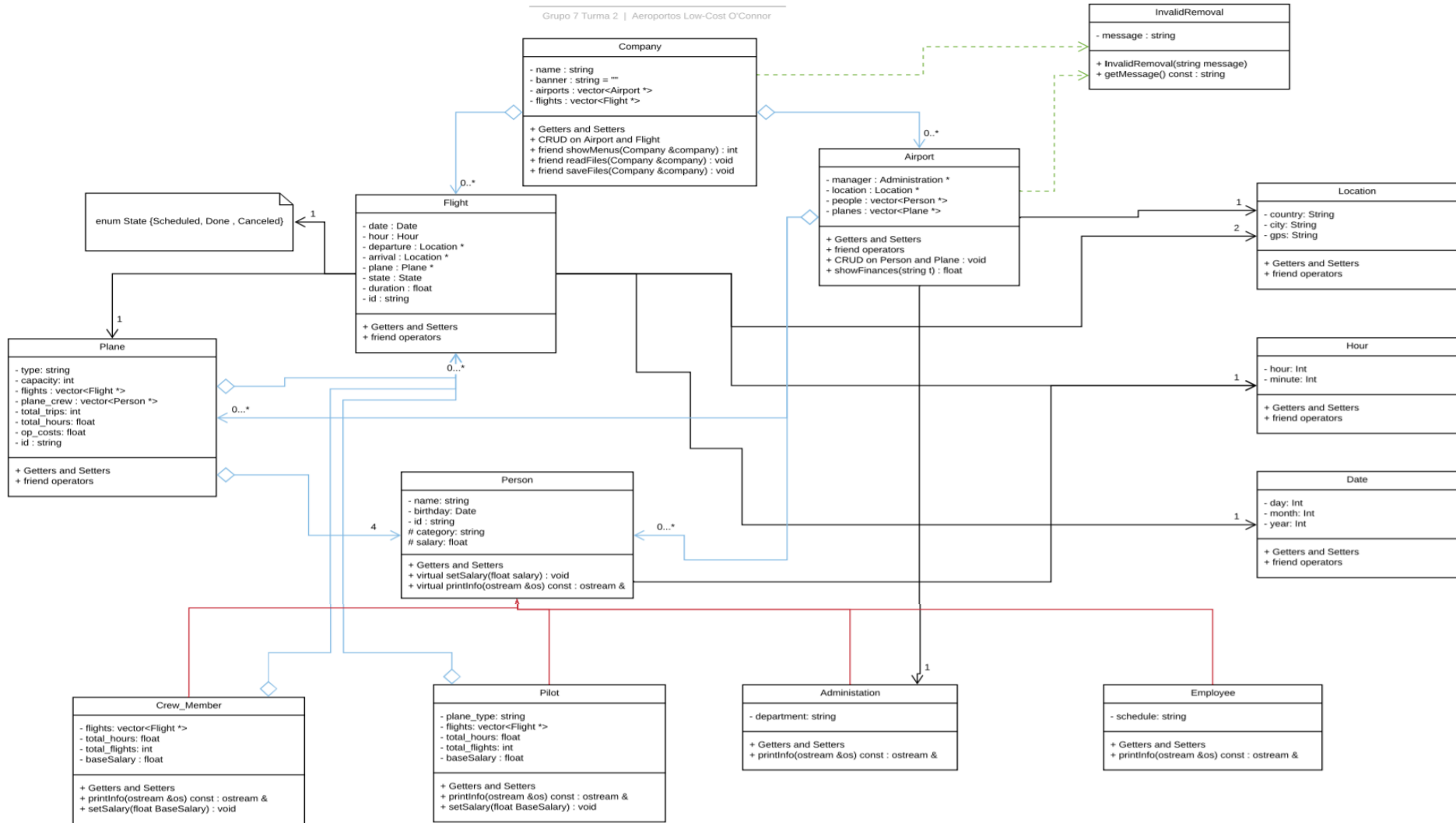


# 02

## Diagrama de Classes

---





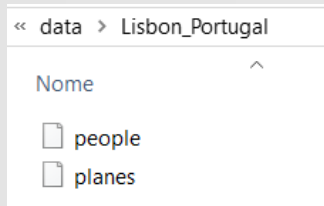
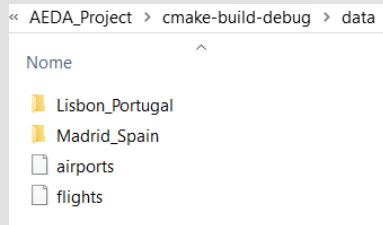


# 03

## Estrutura de Ficheiros

---

# Organização dos Ficheiros



No primeiro nível de armazenamento, todos os aeroportos e voos da companhia se encontram no respetivo ficheiro. No segundo nível, todos os aeroportos têm um diretório reservado para sua informação.

Para cada aeroporto, o seu diretório guarda informação relativa às pessoas e aviões que lhe pertencem. Todos estes ficheiros estão estruturados por linhas de informação, cada uma correspondendo a uma entidade da classe respetiva.

```
void readFiles(Company &company){
    CreateDirectory("data", 0);

    string airports_dir = "data\\airports";
    ifstream airports_dir;

    if (!airports.good() || airports.peek() == ifstream::traits_type::eof() || airports.is_open())
    {
        airports.close();
        std::ofstream output(airports_dir, ios::trunc);
    }

    void saveFiles(Company &company){
        string airports_dir = "data\\airports";
        ofstream airports_file(airports_dir, ios::trunc);
        string res;
        stringstream ss(res);
        for(auto i : company.airports){
            ss << i->getManager()->getName() << ";";
            << i->getManager()->getBirthDay().getYear() << ";";
        }
    }
}
```

```
string line;
while (getline(& airports, & line)){
    stringstream ss(line);
    Administration *manager = new Administration();
    Location *location = new Location();
    Date birth = Date();
    string res;
    getline(& ss, & res, delim: ';');
    manager->setName(res);
}
```

```
Pedro;2000;2;2;1200;Pilot;Boeing 737;0;0
Daniel;2000;2;2;1200;Pilot;Boeing 737;0;0
Joao;2000;1;1;200;Pilot;Airbus A330;0;0
```

Cada linha corresponde a uma entidade e cada parâmetro separado pelo sinal de pontuação " ; " corresponde à informação de um membro dado da respetiva entidade.

## Observação:

A pasta "data", onde é armazenada toda a informação, necessita de existir no mesmo diretório que o executável do programa. Esta, no entanto, é criada vazia, automaticamente, caso ainda não exista informação.

# Detalhes da Organização dos Ficheiros

## Ficheiro people

```
1 Sandra Gomes;1985;8;4;1230.5;Admin;Finances
2 Maria Vaz;1987;5;7;1005.24;Employee;Mondays to Fridays, 17:00 to 01:00
3 Nuno Mendes;1968;12;31;1420.3;Crew;0;0
  Joana Costa;1984;5;8;1756.23;Crew;0;0
  Hugo Mota;1978;11;30;17782.3;Crew;0;0
  Bruno Vasques;1992;1;6;1546.21;Crew;0;0
  Marta Henriques;1980;2;26;1325.99;Crew;0;0
4 Antonio Fernandes;1987;8;6;5462.36;Pilot;Airbus A330;0;0
```

1. Nome, data de nascimento (a;m;d), salário (euros), categoria, departamento
2. Nome, data de nascimento (a;m;d), salário (euros), categoria, horário
3. Nome, data de nascimento (a;m;d), salário (euros), categoria, horas de voo, total de voos
4. Nome, data de nascimento (a;m;d), salário (euros), categoria, tipo de avião, horas de voo, total de voos

## Ficheiro planes

```
Airbus A319;155;885.99;0.75;1;T4P1985;G31P1987;P16C2000;S5C1989
Airbus A320;200;750.5;0;0;F17P1987;F21P1980;P16C2000;J6C1990
```

Tipo de avião, capacidade, custos (euros), horas de voo, voos totais, ids dos membros da tripulação (2 pilotos e 2 assistentes de bordo)

## Ficheiro flights

```
2019;11;12;18;50;0.75;Portugal;Porto;456987;Portugal;Lisbon;145632;1;9155A
2019;11;13;11;27;3;Portugal;Porto;456987;Portugal;Lisbon;145632;0;9155A
2019;11;11;7;31;4;Portugal;Porto;456987;Portugal;Lisbon;145632;0;9155A
2020;1;14;12;20;0.75;Portugal;Porto;456987;Portugal;Lisbon;145632;0;0200A
2020;1;15;6;47;0.75;Portugal;Lisbon;145632;Portugal;Porto;456987;0;0205A
```

Data (a;m;d), Hora (h;m), duração (horas), localização do aeroporto de origem (país; cidade; coordenadas gps), localização do aeroporto de destino (país; cidade; coordenadas GPS), estado do voo, id do avião

## Ficheiro airports

```
Manuel dos Santos;1984;8;4;2000;Admin;Manager;Portugal;Porto;456987
Antonio Fernandes;1965;3;2;1865.99;Admin;Manager;Portugal;Lisbon;145632
```

Nome do administrador, data de nascimento (a;m;d), salário, categoria, departamento, localização do aeroporto (país; cidade; coordenadas GPS)

# 04

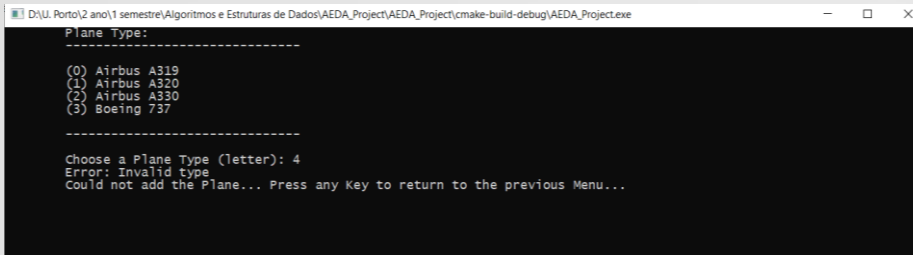
## Tratamento de Exceções

---



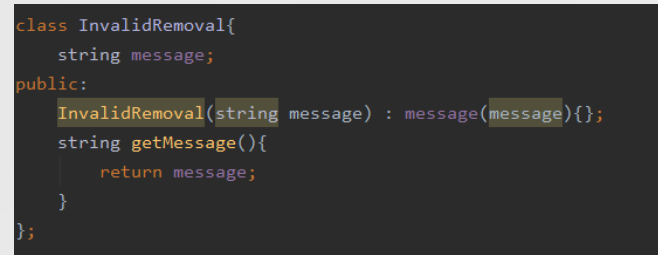
# Exceções

A maioria das exceções lançadas deriva de implementações da STL já existentes. Todos os blocos de código passíveis de gerar exceções estão devidamente sinalizados e as possíveis exceções são tratadas em consequência. Um exemplo é o pedido de input ao utilizador que, quando não corresponde devidamente ao tipo de dado respetivo, gera exceções de vários tipos, sendo estas tratadas nos blocos de catch relacionados. Existem ainda outras exceções criadas para efeitos próprios de organização e flexibilidade do programa. Em baixo são apresentados dois exemplos de exceções e tratamento destas.



```
D:\U.Porto\2 ano\1 semestre\Algoritmos e Estruturas de Dados\AEDA_Project\AEDA_Project\cmake-build-debug\AEDA_Project.exe
Plane Type:
-----
(0) Airbus A319
(1) Airbus A320
(2) Airbus A330
(3) Boeing 737
-----
Choose a Plane Type (letter): 4
Error: Invalid type
Could not add the Plane... Press any Key to return to the previous Menu...
```

Exemplo de exceção que é lançada quando se insere uma opção errada na escolha do tipo de avião.



```
class InvalidRemoval{
    string message;
public:
    InvalidRemoval(string message) : message(message){};
    string getMessage(){
        return message;
    }
};
```

Exemplo da implementação de uma exceção que é chamada quando se tenta remover um piloto/assistente de bordo, quando este está escalado para um determinado voo, ou quando se tenta remover um aeroporto, quando este tem voos planeados.

The background features a complex network of thin grey lines connecting various-sized dark grey circular nodes. These nodes are scattered across the frame, with a higher concentration on the right side, creating a web-like or molecular structure. The overall aesthetic is minimalist and technical.

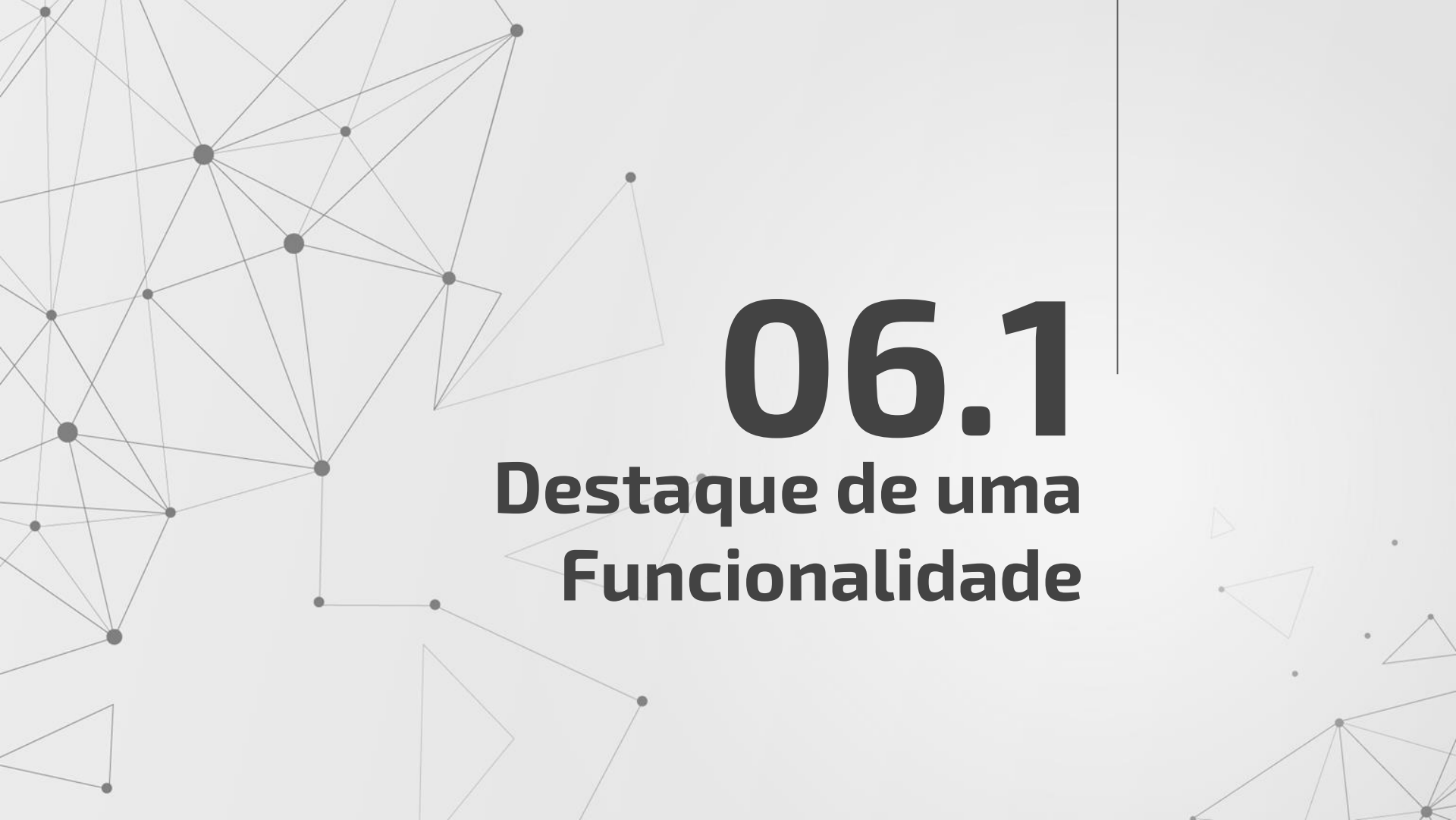
# 05

**Funcionalidades  
Implementadas**

# Lista de funções implementadas

- Criar, editar e remover aeroporto (OK);
- Criar, editar e remover um voo (OK);
- Criar, editar e remover um funcionário de um determinado aeroporto (administrador, empregado, assistente de bordo, piloto) (OK);
- Criar, editar e remover um avião de um aeroporto (OK);
- Listagem de todos os aeroportos (OK);
- Listagem de todos os voos, ordenados por proximidade de data (OK);
- Listagem de todos os trabalhadores de um determinado aeroporto, ordenada por ordem alfabética (OK);
- Listagem de todos os aviões de um determinado aeroporto (OK);
- Atribuição de uma tripulação a um determinado voo (2 pilotos e 2 assistentes de bordo) (OK);
- Atribuição de um voo a um determinado avião (OK);
- Área financeira, geral ou por cada aeroporto (OK);
- Leitura e escrita nos ficheiros (OK);
- Tratamento de exceções (OK);
- Apresentação de uma listagem de todos os voos disponibilizados pela companhia (OK);
- Possibilidade de pesquisar por voos com partida ou chegada num certo aeroporto, entre duas datas ou por estado deste (OK).





# 06.1

## **Destaque de uma Funcionalidade**

# Destaque de uma Funcionalidade

Consideramos merecedor de destaque a implementação do código baseada no mecanismo de apontadores e referências existente na linguagem. Com a estruturação do código feita desta forma, tornou-se muito mais fiável e rápida a implementação das operações de CRUD, já que tornamos a visualização e a atualização do programa instantâneas, após qualquer alteração na informação dos objetos.

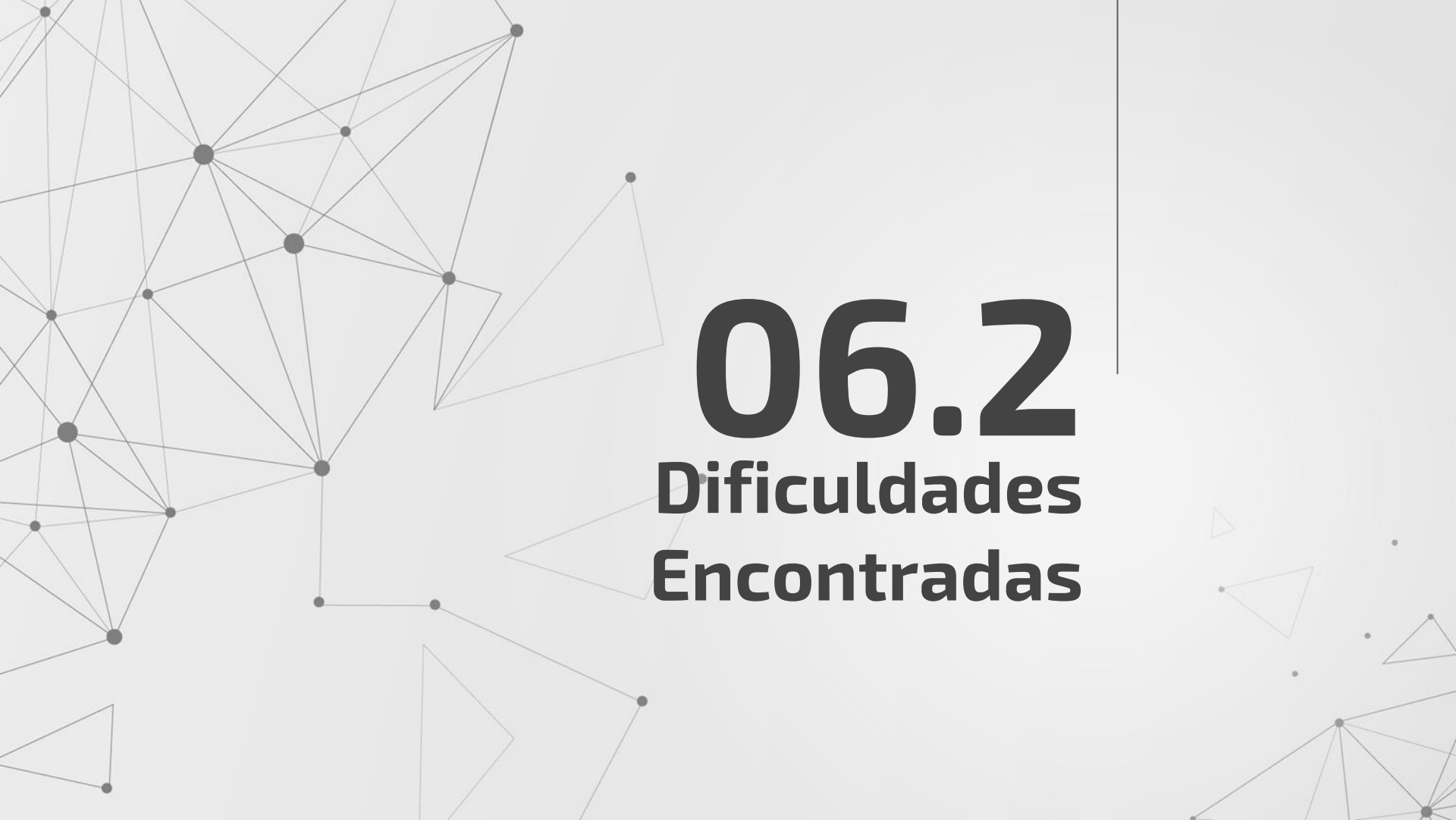
```
switch (opt)
{
    case 'A': {
        flight->setState( state: Scheduled);
        break;
    }
    case 'B': {
        flight->setState( state: Done);
        Plane *plane = flight->getPlane();
        plane->setTotalTrips( totalTrips: plane->getTotalTrips() + 1);
        plane->setTotalHours( totalHours: plane->getTotalHours() + flight->getDuration());
        for(auto person : plane->getPlaneCrew()){
            if(person->getCategory() == "Pilot"){
                Plane_Pilot * planePilot = dynamic_cast<Plane_Pilot *>(person);
                planePilot->setTotalFlights( totalFlights: planePilot->getTotalFlights() + 1);
                planePilot->setTotalHours( totalHours: planePilot->getTotalHours() + flight->getDuration());
            }
            else if(person->getCategory() == "Crew"){
                Crew_Member * crewMember = dynamic_cast<Crew_Member *>(person);
                crewMember->setTotalFlights( totalFlights: crewMember->getTotalFlights() + 1);
                crewMember->setTotalHours( totalHours: crewMember->getTotalHours() + flight->getDuration());
            }
        }
    }
}
```

```
class Airport{
    Administration *manager;
    Location *location;
    vector<Person *> people;
    vector<Plane *> planes;
```

```
class Company{
private:
    string name;
    vector<Airport *> airports = vector<Airport *>();
    vector<Flight *> flights = vector<Flight *>();
    string banner = "";

public:
    Company(const string &name);
    ~Company();

    const string &getName() const { return name; }
```



# 06.2

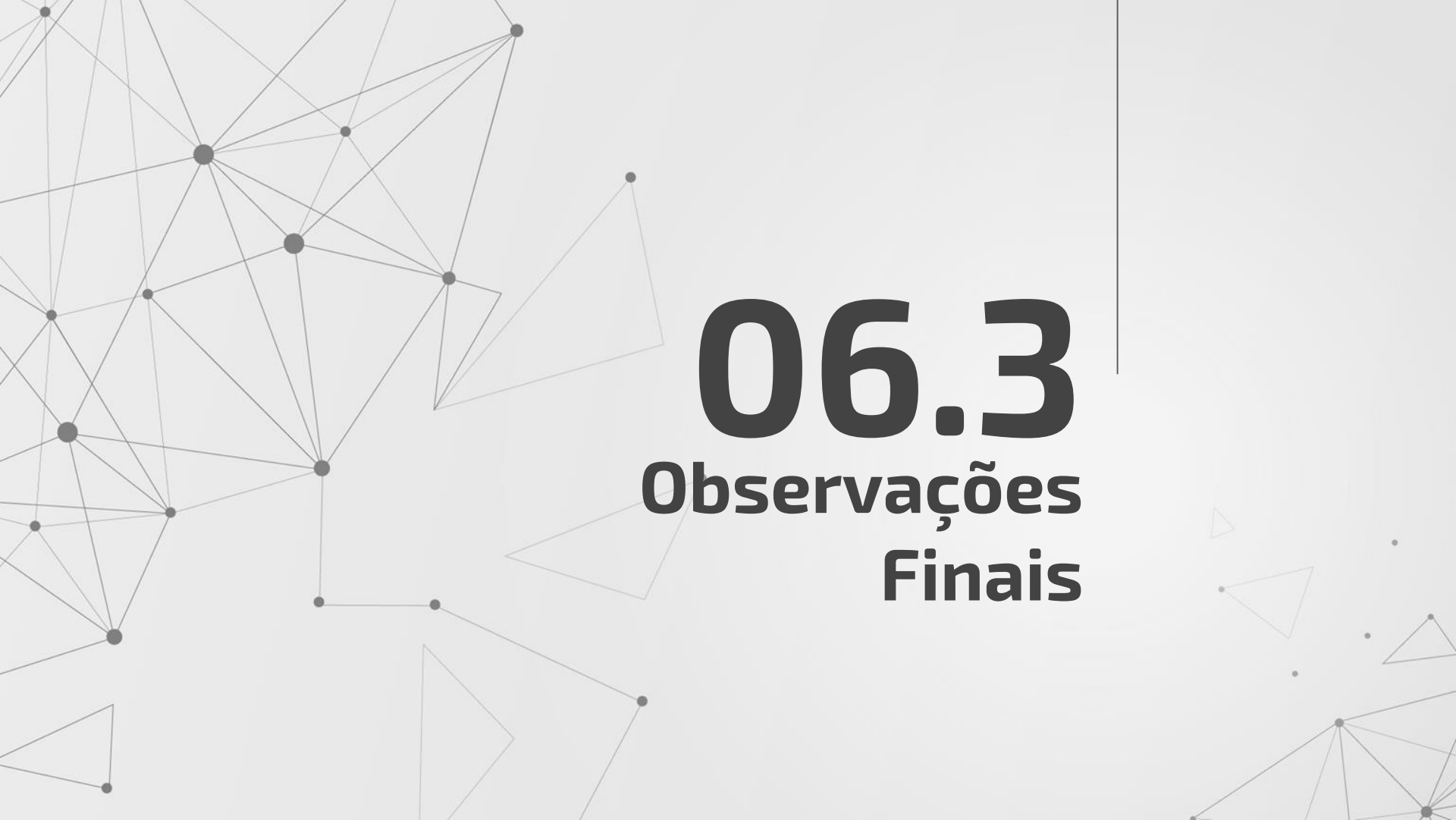
## Dificuldades Encontradas

# Dificuldades Encontradas

Uma das dificuldades que observamos, ao longo do desenvolvimento do projeto, foi a capacidade de ajustamento e materialização de certas informações do enunciado. Vários pontos descritos na ficha foram alterados para poderem ser adaptáveis e verdadeiramente úteis e usáveis no funcionamento geral do programa. Um dos exemplos é a classe Flight que passou a incluir um membro-dado *state* que reflete efetivamente o estado do Voo e um membro-dado *duration* que indica o número de horas do voo - entre outras funcionalidades por nós adicionadas ou alteradas.

Noutros aspetos, tivemos que manipular as entidades de uma forma diferente, pelo mesmo motivo, já que ficávamos impedidos de gerir as dependências das classes, uma vez que todas ficavam interligadas, num loop de “includes” impossível de solucionar. Um exemplo para isso é, na mesma classe Flight, a associação informativa de uma Localização em vez de um Aeroporto de origem e destino – solucionando os conflitos nas classes instanciadas na classe Airport, nomeadamente, as classes derivadas de Person- Plane\_Pilot e Crew\_Member- e a classe Plane que, por sua vez, também dependiam dos Voos.

```
class Flight{  
  
    Date date;  
    Hour hour;  
    Location *departure;  
    Location *arrival;  
    Plane *plane;  
    State state;  
    float duration;  
    string id;
```













# 06.3

## Observações Finais

# Observações Finais

O nosso trabalho foi realizado com a ferramenta Clion, que nos permite gerir as configurações de Run/Debug. Sendo assim, todo o trabalho foi compilado em modo Debug, pelo compilador MinGW, e testado como executável fora do IDE, gerado na pasta cmake-build-debug. Por forma a executar a aplicação, tivemos que garantir que o compilador existia como variável de ambiente, na nossa máquina, e ainda que a pasta “data”, que continha a informação guardada em ficheiros, subsistisse no mesmo diretório que o executável.

Em termos de esforço e de empenho na execução deste projeto, estes foram aproximadamente iguais entre os 3 membros o grupo.

 CMakeFiles	16/11/2019 11:35	Pasta de ficheiros	
 data	08/11/2019 11:43	Pasta de ficheiros	
 AEDA_Project.dbp	16/11/2019 11:35	Ficheiro CBP	11 KB
 AEDA_Project.exe	11/11/2019 12:02	Aplicação	1 444 KB
 cmake_install.cmake	26/10/2019 18:12	Ficheiro CMAKE	2 KB
 CMakeCache.txt	02/11/2019 01:15	Documento de texto	51 KB
 CMakeDoxfile.in	16/11/2019 11:35	Ficheiro IN	15 KB
 CMakeDoxigenDefaults.cmake	16/11/2019 11:35	Ficheiro CMAKE	20 KB
 Doxyfile	02/11/2019 01:15	Ficheiro	113 KB
 Makefile	16/11/2019 11:35	Ficheiro	8 KB



# PARTE 2

Estruturas de dados não lineares



**01**

**Solução e algoritmos  
relevantes**



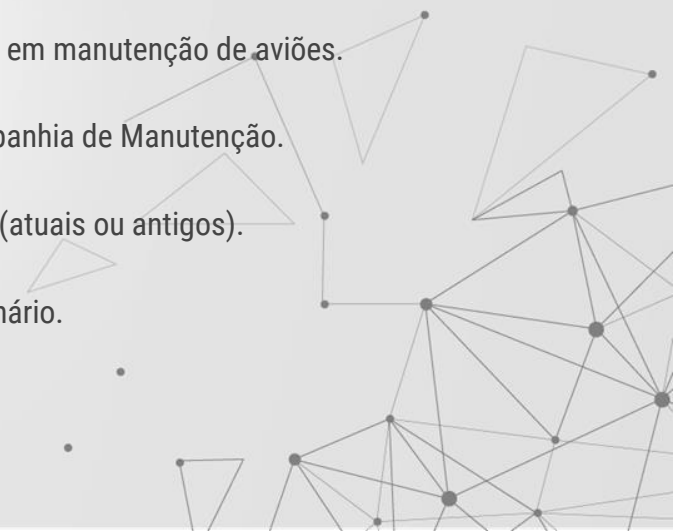
# Descrição do problema

A aplicação a desenvolver faz, nesta segunda parte, uso das estruturas de dados não lineares: árvore binária de pesquisa, fila de prioridade e tabela de dispersão:

1. A O'Connor pretende manter um registo da utilização dos diferentes aeroportos, pelos seus aviões. Para isso, foi criada uma árvore binária de pesquisa para os aeroportos, ordenados pelo número de voos, e, em caso de empate, por tempo médio de permanência em pista. Quando o avião efetua um voo para determinado aeroporto, é atualizada a informação respetiva sobre esse aeroporto. São permitidas listagens várias, tirando partido da ordenação da árvore.
2. Cada avião precisa frequentemente de manutenção de forma a manter os níveis de eficiência definidos e estar de acordo com os critérios de qualidade internacionais. Para tal, a empresa recorre aos serviços de empresas de manutenção especializadas em manutenção de aviões que são guardados numa fila de prioridade ordenada pela disponibilidade das empresas de manutenção. A empresa de manutenção que está no topo é a que estará disponível mais cedo. Para além da disponibilidade, a empresa é também caracterizada pelo número de manutenções já efetuadas.
3. A empresa mantém um registo de todos os seus funcionários (atuais ou antigos) numa tabela de dispersão. A manutenção do registo de funcionários antigos da empresa justifica-se porque, no caso de necessidade de contratação de novos funcionários, a empresa tem como política interna a contratação de funcionários já conhecidos.

# Resolução do problema

Para solucionar o problema atrás descrito, foram criadas as 3 novas estruturas de dados não lineares, juntamente com as classes relacionadas:

1. Árvore de pesquisa para a organização dos registos dos aeroportos.
    - Nova classe AirportRecord com a informação de cada Aeroporto.
  2. Fila de Prioridade para guardar as empresas de manutenção especializadas em manutenção de aviões.
    - Nova classe MaintenanceCompany com a informação de cada Companhia de Manutenção.
  3. Tabela de Dispersão para manter um registo de todos os seus funcionários (atuais ou antigos).
    - Nova classe PersonRecord para o registo do estado de cada Funcionário.
- 

1

```
class Company{
private:
    string name;
    vector<Airport *> airports = vector<Airport *>();
    vector<Flight *> flights = vector<Flight *>();
    set<AirportRecord> AirportRecords;
    priority_queue<MaintenanceCompany> MaintenanceCompanies;
    string banner = "";
};
```

```
class AirportRecord{
    Airport * airport;
    string id;
    int count;
    double track_hours;
    double average_track_time;

public:
    AirportRecord(Airport *airport) : airport(airport) {
        this->id = airport->getLocation()->getUps() + "-" + airport->getManager()->getName();
        this->count = 0;
        this->track_hours = 0;
        this->average_track_time = 0;
    }
};
```

```
bool AirportRecord::operator<(const AirportRecord &rhs) const {
    if (count < rhs.count)
        return true;
    if (rhs.count < count)
        return false;
    return average_track_time < rhs.average_track_time;
}

bool AirportRecord::operator==(const AirportRecord &rhs) const {
    return (airport->getLocation()->getCountry() == rhs.airport->getLocation());
}

bool AirportRecord::operator!=(const AirportRecord &rhs) const {
    return !(*this == *rhs);
}
```

## Árvore de Pesquisa:

Utilizando a classe <set> da stl, foram implementadas as funcionalidades da BST para gerir os registos dos aeroportos, com os respetivos operadores necessários.

2

```
bool operator<(const MaintenanceCompany &rhs) const {
    if (avaiability > rhs.avaiability)
        return true;
    if (rhs.avaiability > avaiability)
        return false;
    return MaintenancesNumber < rhs.MaintenancesNumber;
}
```

## Fila de Prioridade:

Foi usada a implementação da <priority\_queue> da stl para gerir as empresas de manutenção, envolvendo todas as operações de CRUD relacionadas.

3

## Tabela de Dispersão:

Para que todos os funcionários ficassem nos registos dos aeroportos, foi criada uma tabela de dispersão por forma a manter toda a informação dos atuais e antigos trabalhadores dos aeroportos, permitindo empregar funcionários antigos e aceder a todas as suas informações.

```
class PersonRecord {...};

struct peopleHash{
    int operator()(const PersonRecord &p1) const {
        int hash = 7;
        for (int i = 0; i < p1.getPerson()->getName().size(); i++) {
            hash = hash*31 + (int)p1.getPerson()->getName().at(i);
        }
        return hash;
    }

    bool operator()(const PersonRecord &p1, const PersonRecord &p2) const {
        return p1.getPerson()->getId() == p2.getPerson()->getId();
    }
};

typedef unordered_set<PersonRecord, peopleHash, peopleHash> tabHPeople;

class Airport{
    Administration *manager;
    Location *location;
    vector<Person *> people;
    vector<Plane *> planes;
    tabHPeople allWorkers;
};
```

The background features a complex network of thin grey lines connecting various-sized dark grey circular nodes. These nodes are scattered across the frame, with some forming dense clusters and others standing alone. The overall effect is a technical, digital, or network-like aesthetic. A thin vertical line is visible on the left side of the image.

# 02

## Funcionalidades Implementadas

# Lista de funções implementadas

- Criar, editar e remover aeroporto (OK);
  - Criar, editar e remover um voo (OK);
  - Criar, editar e remover um funcionário de um determinado aeroporto (administrador, empregado, assistente de bordo, piloto) (OK);
  - Criar, editar e remover um avião de um aeroporto (OK);
  - Listagem de todos os aeroportos (OK);
  - Listagem de todos os voos, ordenados por proximidade de data (OK);
  - Listagem de todos os trabalhadores de um determinado aeroporto, ordenada por ordem alfabética (OK);
  - Listagem de todos os aviões de um determinado aeroporto (OK);
  - Atribuição de uma tripulação a um determinado voo (2 pilotos e 2 assistentes de bordo) (OK);
  - Atribuição de um voo a um determinado avião (OK);
  - Área financeira, geral ou por cada aeroporto (OK);
  - Leitura e escrita nos ficheiros (OK);
  - Tratamento de exceções (OK);
  - Apresentação de uma listagem de todos os voos disponibilizados pela companhia (OK);
  - Possibilidade de pesquisar por voos com partida ou chegada num certo aeroporto, entre duas datas ou por estado deste(OK);
- 
- Registos dos Aeroportos numa Árvore de Pesquisa (OK);
  - Criar, Editar, Remover e Visualizar os registos dos Aeroportos (OK);
  - Companhias de Manutenção numa Fila de Prioridade (OK);
  - Criar, Editar, Remover e Visualizar as Companhias de Manutenção (OK);
  - Agendar Manutenções e atualizar a fila (OK);
  - Registos dos Funcionários numa Tabela de Dispersão (OK);
  - Criar, Editar, Remover e Visualizar os registos dos funcionários (OK);
  - Empregar um funcionário antigo, baseado no seu registo (OK);

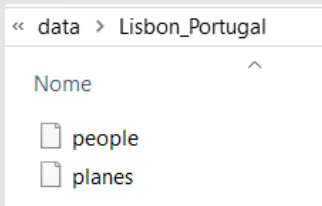
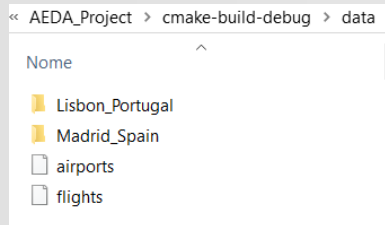


# 03

## Estrutura de Ficheiros

---

# Organização dos Ficheiros



No primeiro nível de armazenamento, todos os aeroportos e voos da companhia se encontram no respetivo ficheiro. No segundo nível, todos os aeroportos têm um diretório reservado para sua informação.

Para cada aeroporto, o seu diretório guarda informação relativa às pessoas e aviões que lhe pertencem. Todos estes ficheiros estão estruturados por linhas de informação, cada uma correspondendo a uma entidade da classe respetiva.

```
void readFiles(Company &company){
    CreatesDirectory("data", &company);

    string airports_dir = "data\\airports";
    ifstream airports(airports_dir);

    if (!airports.good() || airports.peek() == ifstream::traits_type::eof() || airports.is_open())
    {
        airports.close();
        std::ofstream output(airports_dir, ios::trunc);
    }

    void saveFiles(Company &company){
        string airports_dir = "data\\airports";
        ofstream airports_file(airports_dir, ios::trunc);
        string res;
        ostringstream ss(res);
        for(auto i : company.airports){
            ss << i->getManager()->getName() << ";";
            << i->getManager()->getBirthDay().getYear() << ";";
        }
        airports_file << ss.str();
    }
}
```

```
string line;
while (getline(&airports, &line)){
    stringstream ss(line);
    Administration *manager = new Administration();
    Location *location = new Location();
    Date birth = Date();
    string res;
    getline(&ss, &res, delim(';'));
    manager->setName(res);
}
```

Pedro;2000;2;2;1200;Pilot;Boeing 737;0;0  
Daniel;2000;2;2;1200;Pilot;Boeing 737;0;0  
Joao;2000;1;1;200;Pilot;Airbus A330;0;0

Cada linha corresponde a uma entidade e cada parâmetro separado pelo sinal de pontuação ";" corresponde a uma informação de um membro dado da respetiva entidade.

## Observação:

A pasta "data", onde é armazenada toda a informação, necessita de existir no mesmo diretório que o executável do programa. Esta, no entanto, é criada vazia, automaticamente, caso ainda não exista informação.

# Detalhes da Organização dos Ficheiros

## Ficheiro people

```
1 Sandra Gomes;1985;8;4;1230.5;Admin;Finances
2 Maria Vaz;1987;5;7;1005.24;Employee;Mondays to Fridays, 17:00 to 01:00
3 Nuno Mendes;1968;12;31;1420.3;Crew;0;0
  Joana Costa;1984;5;8;1756.23;Crew;0;0
  Hugo Mota;1978;11;30;17782.3;Crew;0;0
  Bruno Vasques;1992;1;6;1546.21;Crew;0;0
  Marta Henriques;1980;2;26;1325.99;Crew;0;0
4 Antonio Fernandes;1987;8;6;5462.36;Pilot;Airbus A330;0;0
```

1. Nome, data de nascimento (a;m;d), salário (euros), categoria, departamento
2. Nome, data de nascimento (a;m;d), salário (euros), categoria, horário
3. Nome, data de nascimento (a;m;d), salário (euros), categoria, horas de voo, total de voos
4. Nome, data de nascimento (a;m;d), salário (euros), categoria, tipo de avião, horas de voo, total de voos

## Ficheiro planes

```
Airbus A319;155;885.99;0.75;1;T4P1985;G31P1987;P16C2000;S5C1989
Airbus A320;200;750.5;0;0;F17P1987;F21P1980;P16C2000;J6C1990
```

Tipo de avião, capacidade, custos (euros), horas de voo, voos totais, ids dos membros da tripulação (2 pilotos e 2 assistentes de bordo)

## Ficheiro flights

```
2019;11;12;18;50;0.75;Portugal;Porto;456987;Portugal;Lisbon;145632;1;9155A
2019;11;13;11;27;3;Portugal;Porto;456987;Portugal;Lisbon;145632;0;9155A
2019;11;11;7;31;4;Portugal;Porto;456987;Portugal;Lisbon;145632;0;9155A
2020;1;14;12;20;0.75;Portugal;Porto;456987;Portugal;Lisbon;145632;0;0200A
2020;1;15;6;47;0.75;Portugal;Lisbon;145632;Portugal;Porto;456987;0;0205A
```

Data (a;m;d), Hora (h;m), duração (horas), localização do aeroporto de origem (país; cidade; coordenadas gps), localização do aeroporto de destino (país; cidade; coordenadas GPS), estado do voo, id do avião

## Ficheiro airports

```
Manuel dos Santos;1984;8;4;2000;Admin;Manager;Portugal;Porto;456987
Antonio Fernandes;1965;3;2;1865.99;Admin;Manager;Portugal;Lisbon;145632
```

Nome do administrador, data de nascimento (a;m;d), salário, categoria, departamento, localização do aeroporto (país; cidade; coordenadas GPS)





# 04.1

## **Destaque de uma Funcionalidade**

# Destaque de uma Funcionalidade

Consideramos merecedor de destaque a implementação do código baseada no mecanismo de apontadores e referências existente na linguagem. Com a estruturação do código feita desta forma, tornou-se muito mais fiável e rápida a implementação das operações de CRUD, já que tornamos a visualização e a atualização do programa instantâneas, após qualquer alteração na informação dos objetos.

```
switch (opt)
{
    case 'A': {
        flight->setState( state: Scheduled);
        break;
    }
    case 'B': {
        flight->setState( state: Done);
        Plane *plane = flight->getPlane();
        plane->setTotalTrips( totalTrips: plane->getTotalTrips() + 1);
        plane->setTotalHours( totalHours: plane->getTotalHours() + flight->getDuration());
        for(auto person : plane->getPlaneCrew()){
            if(person->getCategory() == "Pilot"){
                Plane_Pilot * planePilot = dynamic_cast<Plane_Pilot *>(person);
                planePilot->setTotalFlights( totalFlights: planePilot->getTotalFlights() + 1);
                planePilot->setTotalHours( totalHours: planePilot->getTotalHours() + flight->getDuration());
            }
            else if(person->getCategory() == "Crew"){
                Crew_Member * crewMember = dynamic_cast<Crew_Member *>(person);
                crewMember->setTotalFlights( totalFlights: crewMember->getTotalFlights() + 1);
                crewMember->setTotalHours( totalHours: crewMember->getTotalHours() + flight->getDuration());
            }
        }
    }
}
```

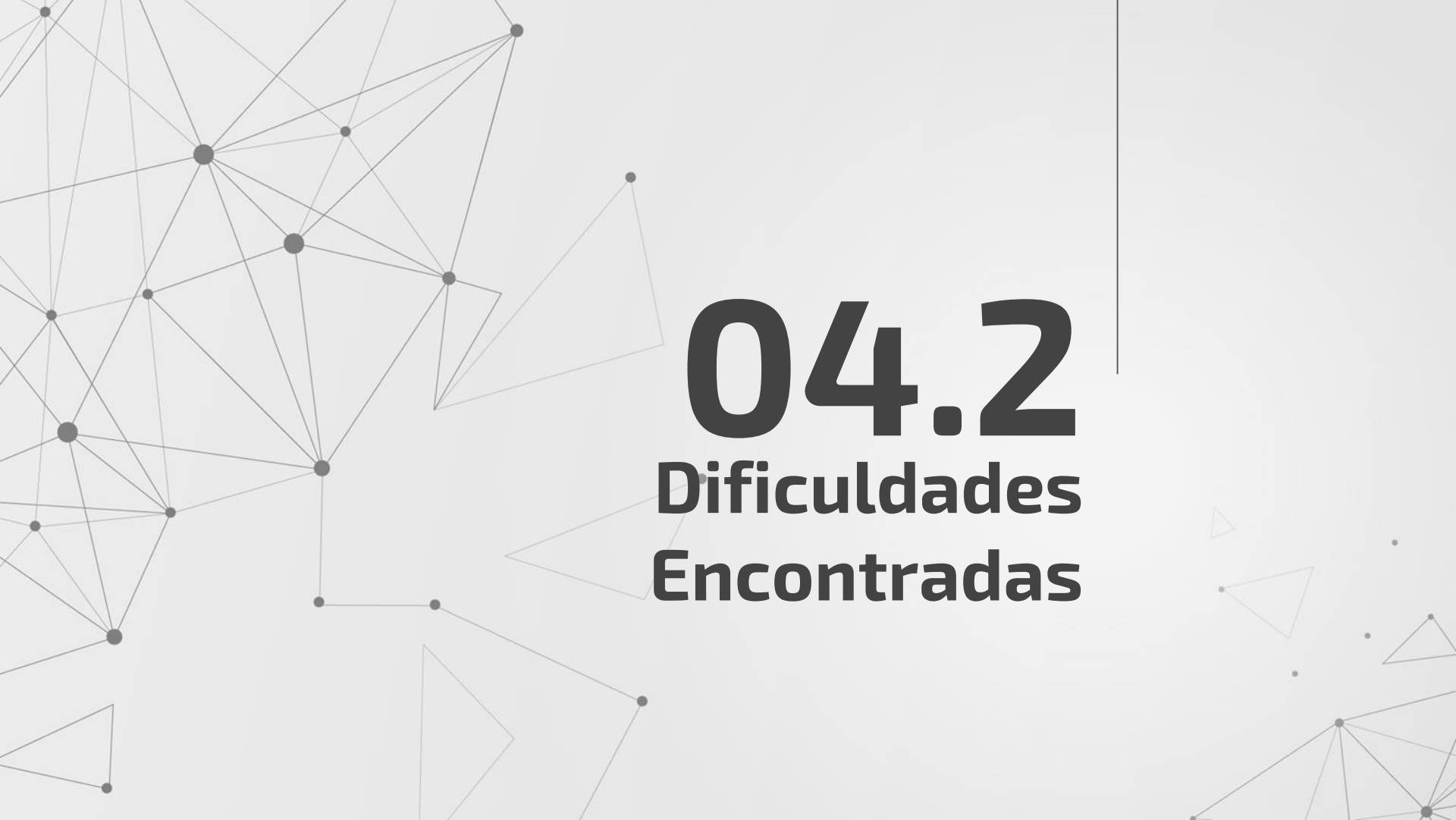
```
class PersonRecord{

    string state;
    Person * person;
```

```
class AirportRecord{

    Airport * airport;
    string id;
    int count;
    double track_hours;
    double average_track_time;

public:
    AirportRecord(Airport *airport) : airport(airport) {
        this->id = airport->getLocation()->getGps() + "." + airport->getManager()->getName();
        this->count = 0;
        this->track_hours = 0;
        this->average_track_time = 0;
    }
}
```



# 04.2

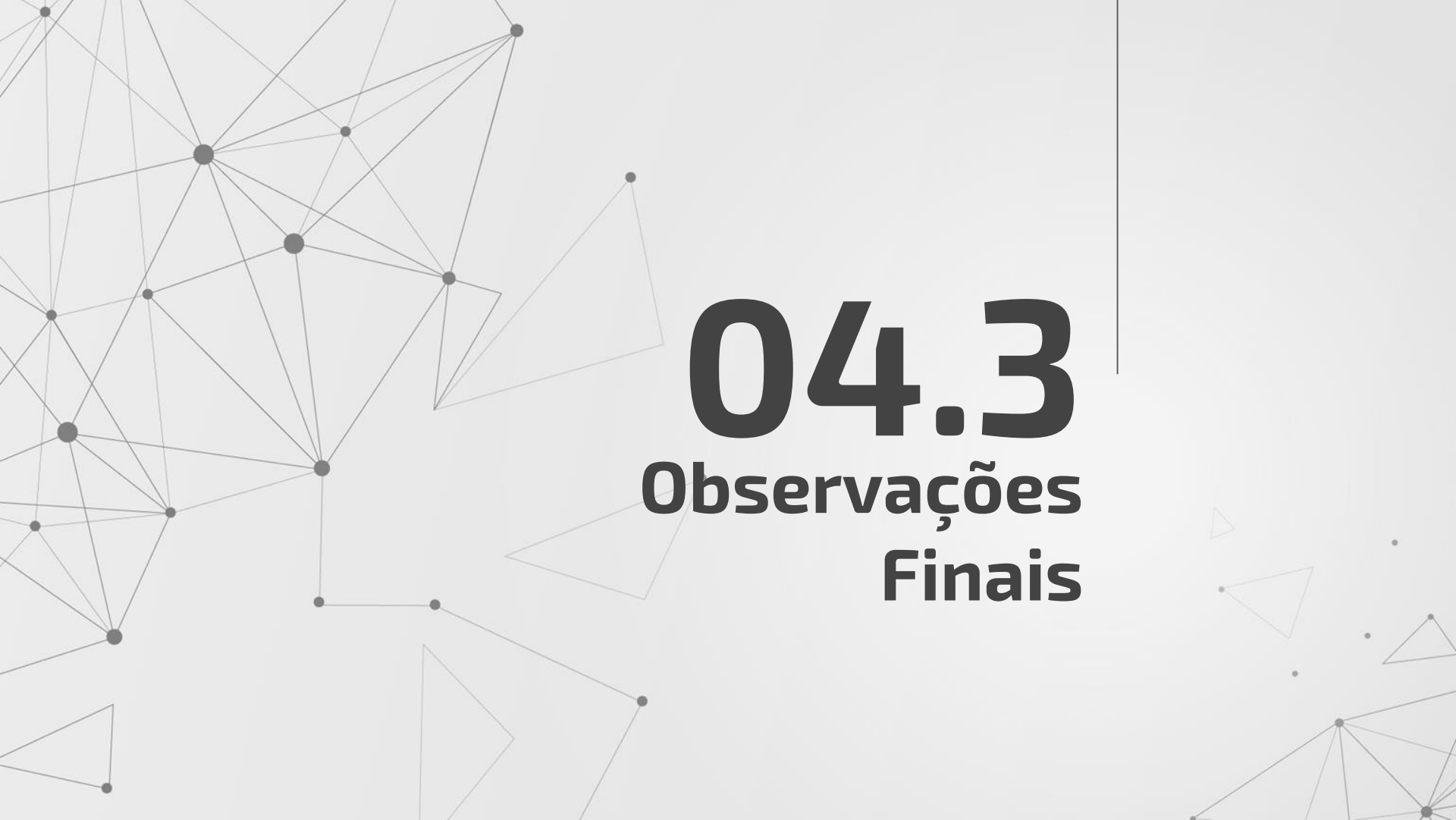
## Dificuldades Encontradas

# Dificuldades Encontradas

Uma das dificuldades que observamos, ao longo do desenvolvimento do projeto, foi a capacidade de ajustamento e materialização de certas informações do enunciado. Vários pontos descritos na ficha foram alterados para poderem ser adaptáveis e verdadeiramente úteis e usáveis no funcionamento geral do programa. Um dos exemplos é a classe Flight que passou a incluir um membro-dado *state* que reflete efetivamente o estado do Voo e um membro-dado *duration* que indica o número de horas do voo - entre outras funcionalidades por nós adicionadas ou alteradas.

Noutros aspetos, tivemos que manipular as entidades de uma forma diferente, pelo mesmo motivo, já que ficávamos impedidos de gerir as dependências das classes, uma vez que todas ficavam interligadas, num loop de “includes” impossível de solucionar. Um exemplo para isso é, na mesma classe Flight, a associação informativa de uma Localização em vez de um Aeroporto de origem e destino – solucionando os conflitos nas classes instanciadas na classe Airport, nomeadamente, as classes derivadas de Person- Plane\_Pilot e Crew\_Member- e a classe Plane que, por sua vez, também dependiam dos Voos.

```
class Flight{  
  
    Date date;  
    Hour hour;  
    Location *departure;  
    Location *arrival;  
    Plane *plane;  
    State state;  
    float duration;  
    string id;|
```













# 04.3

## Observações Finais

# Observações Finais

O nosso trabalho foi realizado com a ferramenta Clion, que nos permite gerir as configurações de Run/Debug. Sendo assim, todo o trabalho foi compilado em modo Debug, pelo compilador MinGW, e testado como executável fora do IDE, gerado na pasta cmake-build-debug. Por forma a executar a aplicação, tivemos que garantir que o compilador existia como variável de ambiente, na nossa máquina, e ainda que a pasta “data”, que continha a informação guardada em ficheiros, subsistisse no mesmo diretório que o executável.

Em termos de esforço e de empenho na execução deste projeto, estes foram aproximadamente iguais entre os 3 membros o grupo.

 CMakeFiles	16/11/2019 11:35	Pasta de ficheiros	
 data	08/11/2019 11:43	Pasta de ficheiros	
 AEDA_Project.dbp	16/11/2019 11:35	Ficheiro CBP	11 KB
 AEDA_Project.exe	11/11/2019 12:02	Aplicação	1 444 KB
 cmake_install.cmake	26/10/2019 18:12	Ficheiro CMAKE	2 KB
 CMakeCache.txt	02/11/2019 01:15	Documento de texto	51 KB
 CMakeDoxfile.in	16/11/2019 11:35	Ficheiro IN	15 KB
 CMakeDoxxygenDefaults.cmake	16/11/2019 11:35	Ficheiro CMAKE	20 KB
 Doxyfile	02/11/2019 01:15	Ficheiro	113 KB
 Makefile	16/11/2019 11:35	Ficheiro	8 KB



# FIM

Aeroportos Low-Cost O'Connor

Eduardo Brito - up201806271  
Pedro Ferreira - up201806506  
Pedro Ponte - up201809694

