FlightPath

1.0

Generated by Doxygen 1.10.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Airline Class Reference

`#include <Airline.h>`

Public Member Functions

- Airline (std::string code, std::string name, std::string callsign, std::string country)
  Constructor for Airline class Complexity: O(1)
- Airline (std::string code)
  Constructor for Airline class Complexity: O(1)
- std::string getCode ()
  Getter for retrieving Airline code Complexity: O(1)
- std::string getName ()
  Getter for retrieving Airline name Complexity: O(1)
- std::string getCallsign ()
  Getter for retrieving Airline callsign Complexity: O(1)
- std::string getCountry ()
  Getter for retrieving Airline country Complexity: O(1)
- void setCode (std::string code)
  Setter for set/modifying the Airline code Complexity: O(1)
- void setName (std::string name)
  Setter for set/modifying the Airline name Complexity: O(1)
- void setCallsign (std::string callsign)
  Setter for set/modifying the Airline callsign Complexity: O(1)
- void setCountry (std::string country)
  Setter for set/modifying the Airline country Complexity: O(1)
- void display ()
  Function to print the Airline information Complexity: O(1)
- bool operator== (const Airline &other) const
  Function for comparing two Airline code Complexity: O(1)

3.1.1 Detailed Description

Definition at line 7 of file Airline.h.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Airline() [1/2]

```
Airline::Airline (
            std::string code,
            std::string name,
            std::string callsign,
            std::string country )
```

Constructor for Airline class Complexity: O(1)

Parameters

code	The code of the airline.
name	The name of the airline.
callsign	The callsign of the airline.
country	The country of the airline.

Definition at line 11 of file Airline.cpp.

```
00013 {
00014   this->code = code;
00015   this->name = name;
00016   this->callsign = callsign;
00017   this->country = country;
00018 }
```

3.1.2.2 Airline() [2/2]

```
Airline::Airline (
            std::string code )
```

Constructor for Airline class Complexity: O(1)

Parameters

code	The code of the airline.

Definition at line 25 of file Airline.cpp.

```
00025 { this->code = code; }
```

3.1.3 Member Function Documentation

3.1.3.1 display()

```
void Airline::display ( )
```

Function to print the Airline information Complexity: O(1)

Returns

void

Definition at line 43 of file Airline.cpp.

```
00044 {
00045   std::cout « "Code: " « this->code « " / ";
00046   std::cout « "Name: " « this->name « " / ";
00047   std::cout « "Callsign: " « this->callsign « " / ";
00048   std::cout « "Country: " « this->country « " / ";
00049 }
```

3.1.3.2 getCallsign()

```
std::string Airline::getCallsign ( )
```

Getter for retrieving Airline callsign Complexity: O(1)

Returns

std::string The callsign of the airline.

Definition at line 102 of file Airline.cpp.

```
00102 { return callsign; }
```

3.1.3.3 getCode()

```
std::string Airline::getCode ( )
```

Getter for retrieving Airline code Complexity: O(1)

Returns

std::string The code of the airline.

Definition at line 88 of file Airline.cpp.

```
00088 { return code; }
```

3.1.3.4 getCountry()

```
std::string Airline::getCountry ( )
```

Getter for retrieving Airline country Complexity: O(1)

Returns

std::string The country of the airline.

Definition at line 109 of file Airline.cpp.

```
00109 { return country; }
```

3.1.3.5 getName()

```
std::string Airline::getName ( )
```

Getter for retrieving Airline name Complexity: O(1)

Returns

std::string The name of the airline.

Definition at line 95 of file Airline.cpp.

```
00095 { return name; }
```

3.1.3.6 operator==()

```
bool Airline::operator== (
            const Airline & other ) const
```

Function for comparing two Airline code Complexity: O(1)

Parameters

other	The other airline.

Returns

bool True if the two airlines have the same code.

Definition at line 33 of file Airline.cpp.

```
00034 {
00035   return this->code == other.code;
00036 }
```

3.1.3.7 setCallsign()

```
void Airline::setCallsign (
            std::string callsign )
```

Setter for set/modifying the Airline callsign Complexity: O(1)

Parameters

callsign	The callsign of the airline.

Returns

void

Definition at line 73 of file Airline.cpp.

```
00073 { this->callsign = callsign; }
```

3.1.3.8 setCode()

```
void Airline::setCode (
            std::string code )
```

Setter for set/modifying the Airline code Complexity: O(1)

Parameters

code	The code of the airline.

Returns

void

Definition at line 57 of file Airline.cpp.

```
00057 { this->code = code; }
```

3.1.3.9 setCountry()

```
void Airline::setCountry (
            std::string country )
```

Setter for set/modifying the Airline country Complexity: O(1)

Parameters

country	The country of the airline.

Returns

void

Definition at line 81 of file Airline.cpp.

```
00081 { this->country = country; }
```

3.1.3.10 setName()

```
void Airline::setName (
            std::string name )
```

Setter for set/modifying the Airline name Complexity: O(1)

Parameters

name	The name of the airline.

Returns

void

Definition at line 65 of file Airline.cpp.

```
00065 { this->name = name; }
```

The documentation for this class was generated from the following files:

- src/classes/Airline.h
- src/classes/Airline.cpp

3.2 Airport Class Reference

`#include <Airport.h>`

Public Member Functions

- Airport (std::string code, std::string name, std::string country, std::string city, double latitude, double longitude)
  Constructor for Airport class Complexity: O(1)
- Airport (std::string code)
  Constructor for Airport class Complexity: O(1)
- std::string getCode ()
  Getter for retrieving Airport code Complexity: O(1)
- std::string getName ()
  Getter for retrieving Airport name Complexity: O(1)
- std::string getCountry ()
  Getter for retrieving Airport country Complexity: O(1)
- std::string getCity ()
  Getter for retrieving Airport city Complexity: O(1)
- double getLatitude ()
  Getter for retrieving Airport latitude Complexity: O(1)
- double getLongitude ()
  Getter for retrieving Airport longitude Complexity: O(1)
- void setCode (std::string code)
  Setter for set/modifying the Airport code Complexity: O(1)
- void setName (std::string name)
  Setter for set/modifying the Airport name Complexity: O(1)
- void setCountry (std::string country)
  Setter for set/modifying the Airport country Complexity: O(1)
- void setCity (std::string city)
  Setter for set/modifying the Airport city Complexity: O(1)
- void setLatitude (double latitude)
  Setter for set/modifying the Airport latitude Complexity: O(1)
- void setLongitude (double longitude)
  Setter for set/modifying the Airport longitude Complexity: O(1)
- void display ()
  Function to print the Aiport information Complexity: O(1)
- bool operator== (const Airport &other) const
  Function for comparing two Airport code Complexity: O(1)

3.2.1 Detailed Description

Definition at line 13 of file Airport.h.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Airport() [1/2]

```
Airport::Airport (
            std::string code,
            std::string name,
            std::string country,
            std::string city,
            double latitude,
            double longitude )
```

Constructor for Airport class Complexity: O(1)

Parameters

code	The code of the airport.
name	The name of the airport.
country	The country of the airport.
city	The city of the airport.
latitude	The latitude of the airport.
longitude	The longitude of the airport.

Definition at line 13 of file Airport.cpp.

```
00015 {
00016   this->code = code;
00017   this->name = name;
00018   this->country = country;
00019   this->city = city;
00020   this->latitude = latitude;
00021   this->longitude = longitude;
00022 }
```

3.2.2.2 Airport() [2/2]

```
Airport::Airport (
            std::string code )
```

Constructor for Airport class Complexity: O(1)

Parameters

code	The code of the airport.

Definition at line 29 of file Airport.cpp.

```
00029 { this->code = code; }
```

3.2.3 Member Function Documentation

3.2.3.1 display()

```
void Airport::display ( )
```

Function to print the Aiport information Complexity: O(1)

Returns

void

Definition at line 62 of file Airport.cpp.

```
00063 {
00064   std::cout « "Code: " « this->code « " / ";
00065   std::cout « "Name: " « this->name « " / ";
00066   std::cout « "Country: " « this->country « " / ";
00067   std::cout « "City: " « this->city « " / ";
00068   std::cout « "Latitude: " « this->latitude « " / ";
00069   std::cout « "Longitude: " « this->longitude « std::endl;
00070 }
```

3.2.3.2 getCity()

```
std::string Airport::getCity ( )
```

Getter for retrieving Airport city Complexity: O(1)

Returns

std::string The city of the airport.

Definition at line 146 of file Airport.cpp.

```
00146 { return city; }
```

3.2.3.3 getCode()

```
std::string Airport::getCode ( )
```

Getter for retrieving Airport code Complexity: O(1)

Returns

std::string The code of the airport.

Definition at line 125 of file Airport.cpp.

```
00125 { return code; }
```

3.2.3.4 getCountry()

```
std::string Airport::getCountry ( )
```

Getter for retrieving Airport country Complexity: O(1)

Returns

std::string The country of the airport.

Definition at line 139 of file Airport.cpp.
```
00139 { return country; }
```

3.2.3.5 getLatitude()

```
double Airport::getLatitude ( )
```

Getter for retrieving Airport latitude Complexity: O(1)

Returns

double The latitude of the airport.

Definition at line 153 of file Airport.cpp.
```
00153 { return latitude; }
```

3.2.3.6 getLongitude()

```
double Airport::getLongitude ( )
```

Getter for retrieving Airport longitude Complexity: O(1)

Returns

double The longitude of the airport.

Definition at line 160 of file Airport.cpp.
```
00160 { return longitude; }
```

3.2.3.7 getName()

```
std::string Airport::getName ( )
```

Getter for retrieving Airport name Complexity: O(1)

Returns

std::string The name of the airport.

Definition at line 132 of file Airport.cpp.
```
00132 { return name; }
```

3.2.3.8 operator==()

```
bool Airport::operator== (
            const Airport & other ) const
```

Function for comparing two Airport code Complexity: O(1)

Parameters

other	The other airport.

Returns

bool True if the two airports have the same code.

Definition at line 52 of file Airport.cpp.

```
00053 {
00054   return this->code == other.code;
00055 }
```

3.2.3.9 setCity()

```
void Airport::setCity (
            std::string city )
```

Setter for set/modifying the Airport city Complexity: O(1)

Parameters

city	The city of the airport.

Returns

void

Definition at line 102 of file Airport.cpp.

```
00102 { this->city = city; }
```

3.2.3.10 setCode()

```
void Airport::setCode (
            std::string code )
```

Setter for set/modifying the Airport code Complexity: O(1)

Parameters

code	The code of the airport.

Returns

void

Definition at line 78 of file Airport.cpp.

```
00078 { this->code = code; }
```

3.2.3.11 setCountry()

```
void Airport::setCountry (
            std::string country )
```

Setter for set/modifying the Airport country Complexity: O(1)

Parameters

country	The country of the airport.

Returns

void

Definition at line 94 of file Airport.cpp.

```
00094 { this->country = country; }
```

3.2.3.12 setLatitude()

```
void Airport::setLatitude (
            double latitude )
```

Setter for set/modifying the Airport latitude Complexity: O(1)

Parameters

latitude	The latitude of the airport.

Returns

void

Definition at line 110 of file Airport.cpp.

```
00110 { this->latitude = latitude; }
```

3.2.3.13 setLongitude()

```
void Airport::setLongitude (
            double longitude )
```

Setter for set/modifying the Airport longitude Complexity: O(1)

Parameters

longitude	The longitude of the airport.

Returns

void

Definition at line 118 of file Airport.cpp.

```
00118 { this->longitude = longitude; }
```

3.2.3.14 setName()

```
void Airport::setName (
            std::string name )
```

Setter for set/modifying the Airport name Complexity: O(1)

Parameters

name	The name of the airport.

Returns

void

Definition at line 86 of file Airport.cpp.

```
00086 { this->name = name; }
```

The documentation for this class was generated from the following files:

- src/classes/Airport.h
- src/classes/Airport.cpp

3.3 Edge< T > Class Template Reference

```
#include <Graph.h>
```

Public Member Functions

- Edge (Vertex< T > *d, double w, std::string route)

  Constructor for Edge class.
- Vertex< T > * getDest () const

  Function to get the destination vertex of an edge.
- double getWeight () const

  Function to get the weight of an edge.
- std::string getRoute () const

  Function to get the route information of an edge.
- void setDest (Vertex< T > *dest)

  Function to set the destination vertex of an edge.
- void setWeight (double weight)

  Function to set the weight of an edge.

Friends

- class Graph< T >
- class Vertex< T >

3.3.1 Detailed Description

template<class T>
class Edge< T >

Definition at line 67 of file Graph.h.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Edge()

```
template<class T >
Edge< T >::Edge (
            Vertex< T > * d,
            double w,
            std::string r )
```

Constructor for Edge class.

Template Parameters

T	The type of the graph

Parameters

d	The destination vertex
w	The weight of the edge
r	The route information of the edge

Definition at line 149 of file Graph.h.

```
00150     : dest(d), weight(w), route(r) {}
```

3.3.3 Member Function Documentation

3.3.3.1 getDest()

```
template<class T >
Vertex< T > * Edge< T >::getDest ( ) const
```

Function to get the destination vertex of an edge.

Template Parameters

T	The type of the graph

Returns

Vertex<T>* The destination vertex of an edge

Definition at line 218 of file Graph.h.

```
00218 { return dest; }
```

3.3.3.2 getRoute()

```
template<class T >
std::string Edge< T >::getRoute ( ) const
```

Function to get the route information of an edge.

Template Parameters

T	The type of the graph

Returns

std::string The route information of an edge

Definition at line 255 of file Graph.h.

```
00255 { return route; }
```

3.3.3.3 getWeight()

```
template<class T >
double Edge< T >::getWeight ( ) const
```

Function to get the weight of an edge.

Template Parameters

T	The type of the graph

Returns

double The weight of an edge

Definition at line 235 of file Graph.h.

```
00235 { return weight; }
```

3.3.3.4 setDest()

```
template<class T >
void Edge< T >::setDest (
            Vertex< T > * d )
```

Function to set the destination vertex of an edge.

Template Parameters

T	The type of the graph

Parameters

d	The destination vertex

Returns

void

Definition at line 227 of file Graph.h.

```
00227 { Edge::dest = d; }
```

3.3.3.5 setWeight()

```
template<class T >
void Edge< T >::setWeight (
            double weight )
```

Function to set the weight of an edge.

Template Parameters

T	The type of the graph

Parameters

weight	The weight of an edge

Returns

void

Definition at line 244 of file Graph.h.

```
00245 {
00246   Edge::weight = weight;
00247 }
```

3.3.4 Friends And Related Symbol Documentation

3.3.4.1 Graph< T >

```
template<class T >
friend class Graph< T >  [friend]
```

Definition at line 84 of file Graph.h.

3.3.4.2 Vertex< T >

```
template<class T >
friend class Vertex< T >  [friend]
```

Definition at line 84 of file Graph.h.

The documentation for this class was generated from the following file:

- src/classes/Graph.h

3.4 Flight Struct Reference

```
#include <dbairport.h>
```

Public Attributes

- std::string code
- std::string airline

3.4.1 Detailed Description

Definition at line 17 of file dbairport.h.

3.4.2 Member Data Documentation

3.4.2.1 airline

```
std::string Flight::airline
```

Definition at line 20 of file dbairport.h.

3.4.2.2 code

```
std::string Flight::code
```

Definition at line 19 of file dbairport.h.

The documentation for this struct was generated from the following file:

- src/database/dbairport.h

3.5 Graph< T > Class Template Reference

```
#include <Graph.h>
```

Public Member Functions

- Vertex< T > * findVertex (const T &in) const
  Function to find a vertex with a given content.
- int getNumVertex () const
  Function to get the number of vertices in the graph.
- bool addVertex (const T &in)
  Function to add a vertex to a graph.
- bool removeVertex (const T &in)
  Function to remove a vertex from a graph.
- bool addEdge (const T &sourc, const T &dest, double w, std::string route)
  Function to add an edge to a graph.
- bool removeEdge (const T &sourc, const T &dest)
  Function to remove an edge from a graph.
- vector< Vertex< T > * > getVertexSet () const
  Function to get the vector of vertices in the graph.
- vector< T > dfs () const
  Function to perform a depth-first search (dfs) in a graph.
- vector< T > dfs (const T &source) const
  Function to perform a depth-first search (dfs) in a graph.
- vector< T > bfs (const T &source) const
  Function to perform a breadth-first search (bfs) in a graph.
- vector< T > topsort () const
  Function to perform a topological sorting of the vertices of a graph.
- bool isDAG () const
  Function to check if a graph is a DAG.

3.5.1 Detailed Description

template<class T>
class Graph< T >

Definition at line 92 of file Graph.h.

3.5.2 Member Function Documentation

3.5.2.1 addEdge()

```
template<class T >
bool Graph< T >::addEdge (
            const T & sourc,
            const T & dest,
            double w,
            std::string route )
```

Function to add an edge to a graph.

Template Parameters

T	The type of the graph

Parameters

sourc	The source vertex
dest	The destination vertex
w	The weight of the edge
route	The route information of the edge

Returns

bool True if the edge was added

Definition at line 390 of file Graph.h.

```
00392 {
00393   auto v1 = findVertex(sourc);
00394   auto v2 = findVertex(dest);
00395   if (v1 == NULL || v2 == NULL)
00396     return false;
00397   v1->addEdge(v2, w, route);
00398   return true;
00399 }
```

3.5.2.2 addVertex()

```
template<class T >
bool Graph< T >::addVertex (
            const T & in )
```

Function to add a vertex to a graph.

Template Parameters

T	The type of the graph

Parameters

in	The content of the vertex

Returns

bool True if the vertex was added

Definition at line 372 of file Graph.h.

```
00373 {
00374   if (findVertex(in) != NULL)
00375     return false;
00376   vertexSet.push_back(new Vertex<T>(in));
00377   return true;
00378 }
```

3.5.2.3 bfs()

```
template<class T >
vector< T > Graph< T >::bfs (
            const T & source ) const
```

Function to perform a breadth-first search (bfs) in a graph.

Template Parameters

T	The type of the graph

Parameters

source	The source vertex

Returns

vector<T> The vector with the contents of the vertices by bfs order

Definition at line 537 of file Graph.h.

```
00538 {
00539   vector<T> res;
00540   auto s = findVertex(source);
00541   if (s == NULL)
00542     return res;
00543   queue<Vertex<T> *> q;
00544   for (auto v : vertexSet)
00545     v->visited = false;
00546   q.push(s);
00547   s->visited = true;
00548   while (!q.empty())
00549   {
00550     auto v = q.front();
00551     q.pop();
00552     res.push_back(v->info);
00553     for (auto &e : v->adj)
00554     {
00555       auto w = e.dest;
00556       if (!w->visited)
00557       {
00558         q.push(w);
00559         w->visited = true;
00560       }
00561     }
00562   }
00563   return res;
00564 }
```

3.5.2.4 dfs() [1/2]

```
template<class T >
vector< T > Graph< T >::dfs ( ) const
```

Function to perform a depth-first search (dfs) in a graph.

Template Parameters

T	The type of the graph

Returns

vector<T> The vector with the contents of the vertices by dfs order

Definition at line 478 of file Graph.h.

```
00479 {
00480   vector<T> res;
00481   for (auto v : vertexSet)
00482     v->visited = false;
00483   for (auto v : vertexSet)
00484     if (!v->visited)
00485       dfsVisit(v, res);
00486   return res;
00487 }
```

3.5.2.5 dfs() [2/2]

```
template<class T >
vector< T > Graph< T >::dfs (
            const T & source ) const
```

Function to perform a depth-first search (dfs) in a graph.

Template Parameters

T	The type of the graph

Parameters

source	The source vertex

Returns

vector<T> The vector with the contents of the vertices by dfs order

Definition at line 516 of file Graph.h.

```
00517 {
00518   vector<T> res;
00519   auto s = findVertex(source);
00520   if (s == nullptr)
00521     return res;
00522
00523   for (auto v : vertexSet)
00524     v->visited = false;
00525
00526   dfsVisit(s, res);
00527   return res;
00528 }
```

3.5.2.6 findVertex()

```
template<class T >
Vertex< T > * Graph< T >::findVertex (
            const T & in ) const
```

Function to find a vertex with a given content.

Template Parameters

T	The type of the graph

Parameters

in	The content of the vertex

Returns

Vertex<T>* The vertex with the given content

Definition at line 264 of file Graph.h.

```
00265 {
00266   for (auto v : vertexSet)
00267     if (v->info == in)
00268       return v;
00269   return NULL;
00270 }
```

3.5.2.7 getNumVertex()

```
template<class T >
int Graph< T >::getNumVertex ( ) const
```

Function to get the number of vertices in the graph.

Template Parameters

T	The type of the graph

Returns

int The number of vertices in the graph

Definition at line 158 of file Graph.h.

```
00159 {
00160   return vertexSet.size();
00161 }
```

3.5.2.8 getVertexSet()

```
template<class T >
vector< Vertex< T > * > Graph< T >::getVertexSet ( ) const
```

Function to get the vector of vertices in the graph.

Template Parameters

T	The type of the graph

Returns

vector<Vertex<T>*> The vector of vertices in the graph

Definition at line 170 of file Graph.h.

```
00171 {
00172   return vertexSet;
00173 }
```

3.5.2.9 isDAG()

```
template<class T >
bool Graph< T >::isDAG ( ) const
```

Function to check if a graph is a DAG.

Template Parameters

T	The type of the graph

Returns

bool True if the graph is a DAG

Definition at line 572 of file Graph.h.

```
00573 {
00574   for (auto v : vertexSet)
00575   {
00576     v->visited = false;
00577     v->processing = false;
00578   }
00579   for (auto v : vertexSet)
00580     if (!v->visited)
00581       if (!dfsIsDAG(v))
00582         return false;
00583   return true;
00584 }
```

3.5.2.10 removeEdge()

```
template<class T >
bool Graph< T >::removeEdge (
            const T & sourc,
            const T & dest )
```

Function to remove an edge from a graph.

Template Parameters

T	The type of the graph

Parameters

sourc	The source vertex
dest	The destination vertex

Returns

bool True if the edge was removed

Definition at line 423 of file Graph.h.

```
00424 {
00425   auto v1 = findVertex(sourc);
00426   auto v2 = findVertex(dest);
00427   if (v1 == NULL || v2 == NULL)
00428     return false;
00429   return v1->removeEdgeTo(v2);
00430 }
```

3.5.2.11 removeVertex()

```
template<class T >
bool Graph< T >::removeVertex (
            const T & in )
```

Function to remove a vertex from a graph.

Template Parameters

T	The type of the graph

Parameters

in	The content of the vertex to remove

Returns

bool True if the vertex was removed

Definition at line 457 of file Graph.h.

```
00458 {
00459   for (auto it = vertexSet.begin(); it != vertexSet.end(); it++)
00460     if ((*it)->info == in)
00461     {
00462       auto v = *it;
00463       vertexSet.erase(it);
00464       for (auto u : vertexSet)
00465         u->removeEdgeTo(v);
00466       delete v;
00467       return true;
00468     }
00469   return false;
00470 }
```

3.5.2.12 topsort()

```
template<class T >
vector< T > Graph< T >::topsort ( ) const
```

Function to perform a topological sorting of the vertices of a graph.

Template Parameters

T	The type of the graph

Returns

vector<T> The vector with the contents of the vertices by topological order

Definition at line 616 of file Graph.h.

```
00617 {
00618   vector<T> res;
00619
00620   for (auto v : vertexSet)
00621   {
00622     v->indegree = 0;
00623   }
00624
00625   for (auto v : vertexSet)
00626   {
00627     for (auto &e : v->adj)
00628     {
00629       e.dest->indegree += 1;
00630     }
00631   }
00632
00633   std::queue<Vertex<T> *> q;
00634
00635   for (auto v : vertexSet)
00636   {
00637     if (v->indegree == 0)
00638     {
00639       q.push(v);
00640     }
00641   }
00642
00643   while (!q.empty())
00644   {
00645     auto u = q.front();
00646     res.push_back(u->info);
00647     q.pop();
00648
00649     for (auto &e : u->adj)
00650     {
00651       auto w = e.dest;
00652       w->indegree -= 1;
00653       if (w->indegree == 0)
00654       {
00655         q.push(w);
00656       }
00657     }
00658   }
00659   return res;
00660 }
```

The documentation for this class was generated from the following file:

- src/classes/Graph.h

3.6 Ranking Struct Reference

`#include <Airport.h>`

Public Attributes

- std::string code
- int count

3.6.1 Detailed Description

Definition at line 7 of file Airport.h.

3.6.2 Member Data Documentation

3.6.2.1 code

```
std::string Ranking::code
```

Definition at line 8 of file Airport.h.

3.6.2.2 count

```
int Ranking::count
```

Definition at line 9 of file Airport.h.

The documentation for this struct was generated from the following file:

- src/classes/Airport.h

3.7 Vertex< T > Class Template Reference

```
#include <Graph.h>
```

Public Member Functions

- Vertex (T in)
  Constructor for Vertex class.
- int getIndegree () const
  Function to get the indegree of a vertex.
- int getNum () const
  Function to get the numerical identifier of a vertex.
- int getLow () const
  Function to get the low value of a vertex.
- T getInfo () const
  Function to get the content of a vertex.
- bool isVisited () const
  Function to get the visited state of a vertex.
- bool isProcessing () const
  Function to get the processing state of a vertex.
- const vector< Edge< T > > & getAdj () const
  Function to get the vector of adjacent edges of a vertex.
- void setIndegree (int indegree)
  Function to set the indegree of a vertex.

- void setNum (int num)

  Function to set the numerical identifier of a vertex.
- void setLow (int low)

  Function to set the low value of a vertex.
- void setInfo (T in)

  Function to set the content of a vertex.
- void setVisited (bool v)

  Function to set the visited state of a vertex.
- void setProcessing (bool p)

  Function to set the processing state of a vertex.
- void setAdj (const vector< Edge< T > > &adj)

  Function to set the vector of adjacent edges of a vertex.

Friends

- class Graph< T >

3.7.1 Detailed Description

template<class T>
class Vertex< T >

Definition at line 26 of file Graph.h.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 Vertex()

```
template<class T >
Vertex< T >::Vertex (
            T in )
```

Constructor for Vertex class.

Template Parameters

T	The type of the graph

Parameters

in	The content of the vertex

Definition at line 139 of file Graph.h.

```
00139 : info(in) {}
```

3.7.3 Member Function Documentation

3.7.3.1 getAdj()

```
template<class T >
const vector< Edge< T > > & Vertex< T >::getAdj ( ) const
```

Function to get the vector of adjacent edges of a vertex.

Template Parameters

T	The type of the graph

Returns

const vector<Edge<T>>& The vector of adjacent edges

Definition at line 349 of file Graph.h.

```
00350 {
00351   return adj;
00352 }
```

3.7.3.2 getIndegree()

```
template<class T >
int Vertex< T >::getIndegree ( ) const
```

Function to get the indegree of a vertex.

Template Parameters

T	The type of the graph

Returns

int The indegree of a vertex

Definition at line 286 of file Graph.h.

```
00286 { return indegree; }
```

3.7.3.3 getInfo()

```
template<class T >
T Vertex< T >::getInfo ( ) const
```

Function to get the content of a vertex.

Template Parameters

T	The type of the graph

Returns

T The content of a vertex

Definition at line 181 of file Graph.h.

```
00181 { return info; }
```

3.7.3.4 getLow()

```
template<class T >
int Vertex< T >::getLow ( ) const
```

Function to get the low value of a vertex.

Template Parameters

T	The type of the graph

Returns

int The low value of a vertex

Definition at line 323 of file Graph.h.

```
00323 { return low; }
```

3.7.3.5 getNum()

```
template<class T >
int Vertex< T >::getNum ( ) const
```

Function to get the numerical identifier of a vertex.

Template Parameters

T	The type of the graph

Returns

int The numerical identifier of a vertex

Definition at line 306 of file Graph.h.

```
00306 { return num; }
```

3.7.3.6 isProcessing()

```
template<class T >
bool Vertex< T >::isProcessing ( ) const
```

Function to get the processing state of a vertex.

Template Parameters

T	The type of the graph

Returns

bool The processing state of a vertex

Definition at line 198 of file Graph.h.

```
00198 { return processing; }
```

3.7.3.7 isVisited()

```
template<class T >
bool Vertex< T >::isVisited ( ) const
```

Function to get the visited state of a vertex.

Template Parameters

T	The type of the graph

Returns

bool The visited state of a vertex

Definition at line 278 of file Graph.h.

```
00278 { return visited; }
```

3.7.3.8 setAdj()

```
template<class T >
void Vertex< T >::setAdj (
            const vector< Edge< T > > & adj )
```

Function to set the vector of adjacent edges of a vertex.

Parameters

adj	The vector of adjacent edges

Returns

void

Definition at line 360 of file Graph.h.

```
00361 {
00362   Vertex::adj = adj;
00363 }
```

3.7.3.9 setIndegree()

```
template<class T >
void Vertex< T >::setIndegree (
             int indegree )
```

Function to set the indegree of a vertex.

Template Parameters

T	The type of the graph

Parameters

indegree	The indegree of a vertex

Returns

void

Definition at line 295 of file Graph.h.

```
00296 {
00297   Vertex::indegree = indegree;
00298 }
```

3.7.3.10 setInfo()

```
template<class T >
void Vertex< T >::setInfo (
             T in )
```

Function to set the content of a vertex.

Template Parameters

T	The type of the graph

Parameters

in	The content of a vertex

Returns

void

Definition at line 190 of file Graph.h.

```
00190 { Vertex::info = in; }
```

3.7.3.11 setLow()

```
template<class T >
void Vertex< T >::setLow (
            int low )
```

Function to set the low value of a vertex.

Template Parameters

T	The type of the graph

Parameters

low	The low value of a vertex

Returns

void

Definition at line 332 of file Graph.h.

```
00332 { Vertex::low = low; }
```

3.7.3.12 setNum()

```
template<class T >
void Vertex< T >::setNum (
            int num )
```

Function to set the numerical identifier of a vertex.

Template Parameters

T	The type of the graph

Parameters

num	The numerical identifier of a vertex

Returns

void

Definition at line 315 of file Graph.h.

```
00315 { Vertex::num = num; }
```

3.7.3.13 setProcessing()

```
template<class T >
void Vertex< T >::setProcessing (
            bool p )
```

Function to set the processing state of a vertex.

Template Parameters

T	The type of the graph

Parameters

p	The processing state of a vertex

Returns

void

Definition at line 207 of file Graph.h.

```
00208 {
00209   Vertex::processing = p;
00210 }
```

3.7.3.14 setVisited()

```
template<class T >
void Vertex< T >::setVisited (
            bool v )
```

Function to set the visited state of a vertex.

Template Parameters

T	The type of the graph

Parameters

v	The visited state of a vertex

Returns

void

Definition at line 341 of file Graph.h.

```
00341 { Vertex::visited = v; }
```

3.7.4 Friends And Related Symbol Documentation

3.7.4.1 Graph< T >

```
template<class T >
friend class Graph< T >  [friend]
```

Definition at line 60 of file Graph.h.

The documentation for this class was generated from the following file:

- src/classes/Graph.h

Chapter 4

File Documentation

4.1 src/classes/Airline.cpp File Reference

```
#include "Airline.h"
```

4.2 Airline.cpp

Go to the documentation of this file.

```
00001 #include "Airline.h"
00002
00011 Airline::Airline(std::string code, std::string name, std::string callsign,
00012                  std::string country)
00013 {
00014   this->code = code;
00015   this->name = name;
00016   this->callsign = callsign;
00017   this->country = country;
00018 }
00019
00025 Airline::Airline(std::string code) { this->code = code; }
00026
00033 bool Airline::operator==(const Airline &other) const
00034 {
00035   return this->code == other.code;
00036 }
00037
00043 void Airline::display()
00044 {
00045   std::cout « "Code: " « this->code « " / ";
00046   std::cout « "Name: " « this->name « " / ";
00047   std::cout « "Callsign: " « this->callsign « " / ";
00048   std::cout « "Country: " « this->country « " / ";
00049 }
00050
00057 void Airline::setCode(std::string code) { this->code = code; }
00058
00065 void Airline::setName(std::string name) { this->name = name; }
00066
00073 void Airline::setCallsign(std::string callsign) { this->callsign = callsign; }
00074
00081 void Airline::setCountry(std::string country) { this->country = country; }
00082
00088 std::string Airline::getCode() { return code; }
00089
00095 std::string Airline::getName() { return name; }
00096
00102 std::string Airline::getCallsign() { return callsign; }
00103
00109 std::string Airline::getCountry() { return country; }
```

4.3 src/classes/Airline.h File Reference

```
#include <iostream>
#include <string>
```

Classes

- class Airline

4.4 Airline.h

Go to the documentation of this file.

```
00001 #ifndef AIRLINE_H
00002 #define AIRLINE_H
00003
00004 #include <iostream>
00005 #include <string>
00006
00007 class Airline {
00008
00009 public:
00010   // Constructor
00011   Airline(std::string code, std::string name, std::string callsign,
00012           std::string country);
00013   Airline(std::string code);
00014
00015   // Getters
00016   std::string getCode();
00017   std::string getName();
00018   std::string getCallsign();
00019   std::string getCountry();
00020
00021   // Setters
00022   void setCode(std::string code);
00023   void setName(std::string name);
00024   void setCallsign(std::string callsign);
00025   void setCountry(std::string country);
00026
00027   // Other functions
00028   void display();
00029   bool operator==(const Airline &other) const;
00030
00031 private:
00032   std::string code;
00033   std::string name;
00034   std::string callsign;
00035   std::string country;
00036 };
00037
00038 #endif
```

4.5 src/classes/Airport.cpp File Reference

```
#include "Airport.h"
```

Functions

- bool comparator (const Ranking a, const Ranking b)

  Function for sorting Airport ranking Complexity: O(1)

4.5.1 Function Documentation

4.5.1.1 comparator()

```
bool comparator (
            const Ranking a,
            const Ranking b )
```

Function for sorting Airport ranking Complexity: O(1)

Parameters

a	The first ranking.
b	The second ranking.

Returns

bool True if the first ranking is greater than the second ranking.

Definition at line 38 of file Airport.cpp.

```
00039 {
00040   if (b.count != a.count)
00041     return a.count > b.count;
00042   else
00043     return a.code < b.code;
00044 }
```

4.6 Airport.cpp

Go to the documentation of this file.

```
00001 #include "Airport.h"
00002
00013 Airport::Airport(std::string code, std::string name, std::string country,
00014                  std::string city, double latitude, double longitude)
00015 {
00016   this->code = code;
00017   this->name = name;
00018   this->country = country;
00019   this->city = city;
00020   this->latitude = latitude;
00021   this->longitude = longitude;
00022 }
00023
00029 Airport::Airport(std::string code) { this->code = code; }
00030
00038 bool comparator(const Ranking a, const Ranking b)
00039 {
00040   if (b.count != a.count)
00041     return a.count > b.count;
00042   else
00043     return a.code < b.code;
00044 }
00045
00052 bool Airport::operator==(const Airport &other) const
00053 {
00054   return this->code == other.code;
00055 }
00056
00062 void Airport::display()
00063 {
00064   std::cout « "Code: " « this->code « " / ";
00065   std::cout « "Name: " « this->name « " / ";
00066   std::cout « "Country: " « this->country « " / ";
00067   std::cout « "City: " « this->city « " / ";
00068   std::cout « "Latitude: " « this->latitude « " / ";
00069   std::cout « "Longitude: " « this->longitude « std::endl;
```

```
00070 }
00071
00078 void Airport::setCode(std::string code) { this->code = code; }
00079
00086 void Airport::setName(std::string name) { this->name = name; }
00087
00094 void Airport::setCountry(std::string country) { this->country = country; }
00095
00102 void Airport::setCity(std::string city) { this->city = city; }
00103
00110 void Airport::setLatitude(double latitude) { this->latitude = latitude; }
00111
00118 void Airport::setLongitude(double longitude) { this->longitude = longitude; }
00119
00125 std::string Airport::getCode() { return code; }
00126
00132 std::string Airport::getName() { return name; }
00133
00139 std::string Airport::getCountry() { return country; }
00140
00146 std::string Airport::getCity() { return city; }
00147
00153 double Airport::getLatitude() { return latitude; }
00154
00160 double Airport::getLongitude() { return longitude; }
```

4.7 src/classes/Airport.h File Reference

```
#include <iostream>
#include <string>
```

Classes

- struct Ranking
- class Airport

Functions

- bool comparator (const Ranking a, const Ranking b)

  Function for sorting Airport ranking Complexity: O(1)

4.7.1 Function Documentation

4.7.1.1 comparator()

```
bool comparator (
            const Ranking a,
            const Ranking b )
```

Function for sorting Airport ranking Complexity: O(1)

Parameters

a	The first ranking.
b	The second ranking.

Returns

bool True if the first ranking is greater than the second ranking.

Definition at line 38 of file Airport.cpp.

```
00039 {
00040   if (b.count != a.count)
00041     return a.count > b.count;
00042   else
00043     return a.code < b.code;
00044 }
```

4.8 Airport.h

Go to the documentation of this file.

```
00001 #ifndef AIRPORT_H
00002 #define AIRPORT_H
00003
00004 #include <iostream>
00005 #include <string>
00006
00007 struct Ranking {
00008   std::string code;
00009   int count;
00010 };
00011
00012 bool comparator(const Ranking a, const Ranking b);
00013 class Airport {
00014
00015 public:
00016   // Constructor
00017   Airport(std::string code, std::string name, std::string country,
00018           std::string city, double latitude, double longitude);
00019   Airport(std::string code);
00020
00021   // Getters
00022   std::string getCode();
00023   std::string getName();
00024   std::string getCountry();
00025   std::string getCity();
00026   double getLatitude();
00027   double getLongitude();
00028
00029   // Setters
00030   void setCode(std::string code);
00031   void setName(std::string name);
00032   void setCountry(std::string country);
00033   void setCity(std::string city);
00034   void setLatitude(double latitude);
00035   void setLongitude(double longitude);
00036
00037   // Other functions
00038   void display();
00039   bool operator==(const Airport &other) const;
00040
00041 private:
00042   std::string code;
00043   std::string name;
00044   std::string country;
00045   std::string city;
00046   double latitude;
00047   double longitude;
00048 };
00049
00050 #endif
```

4.9 src/classes/Graph.h File Reference

```
#include <cstddef>
#include <list>
#include <queue>
#include <stack>
#include <string>
#include <vector>
```

Classes

- class Vertex< T >
- class Edge< T >
- class Graph< T >

4.10 Graph.h

Go to the documentation of this file.

```
00001 /*
00002  * Graph.h
00003  */
00004 #ifndef GRAPH_H_
00005 #define GRAPH_H_
00006
00007 #include <cstddef>
00008 #include <list>
00009 #include <queue>
00010 #include <stack>
00011 #include <string>
00012 #include <vector>
00013
00014 using namespace std;
00015
00016 template <class T>
00017 class Edge;
00018 template <class T>
00019 class Graph;
00020 template <class T>
00021 class Vertex;
00022
00023 /****************** Provided structures  ********************/
00024
00025 template <class T>
00026 class Vertex
00027 {
00028   T info;              // contents
00029   vector<Edge<T>> adj; // list of outgoing edges
00030   bool visited;        // auxiliary field
00031   bool processing;     // auxiliary field
00032   int indegree;        // auxiliary field
00033   int num;             // auxiliary field
00034   int low;             // auxiliary field
00035
00036   // Auxiliar Functions: edge operations
00037   void addEdge(Vertex<T> *dest, double w, std::string route);
00038   bool removeEdgeTo(Vertex<T> *d);
00039
00040 public:
00041   // Constructor
00042   Vertex(T in);
00043
00044   // Getters
00045   int getIndegree() const;
00046   int getNum() const;
00047   int getLow() const;
00048   T getInfo() const;
00049   bool isVisited() const;
00050   bool isProcessing() const;
00051   const vector<Edge<T>> &getAdj() const;
00052
00053   // Setters
00054   void setIndegree(int indegree);
00055   void setNum(int num);
00056   void setLow(int low);
00057   void setInfo(T in);
00058   void setVisited(bool v);
00059   void setProcessing(bool p);
00060   void setAdj(const vector<Edge<T>> &adj);
00061
00062   // Friend class declaration
00063   friend class Graph<T>;
00064 };
00065
00066 template <class T>
00067 class Edge
00068 {
00069   Vertex<T> *dest;   // destination vertex
00070   double weight;     // auxiliary field
```

```
00071   std::string route; // auxiliary field
00072
00073 public:
00074   // Constructor
00075   Edge(Vertex<T> *d, double w, std::string route);
00076
00077   // Getters
00078   Vertex<T> *getDest() const;
00079   double getWeight() const;
00080   std::string getRoute() const;
00081
00082   // Setters
00083   void setDest(Vertex<T> *dest);
00084   void setWeight(double weight);
00085
00086   // Friend class declaration
00087   friend class Graph<T>;
00088   friend class Vertex<T>;
00089 };
00090
00091 template <class T>
00092 class Graph
00093 {
00094   vector<Vertex<T> *> vertexSet; // vertex set
00095   int _index_;                   // auxiliary field
00096   stack<Vertex<T» _stack_;      // auxiliary field
00097   list<list<T» _list_sccs_;     // auxiliary field
00098
00099   // Auxiliar Functions: Search depth-first traversal
00100   void dfsVisit(Vertex<T> *v, vector<T> &res) const;
00101
00102   // Auxiliar Functions: Check if graph is DAG
00103   bool dfsIsDAG(Vertex<T> *v) const;
00104
00105 public:
00106   // Vertex operations
00107   Vertex<T> *findVertex(const T &in) const;
00108   int getNumVertex() const;
00109   bool addVertex(const T &in);
00110   bool removeVertex(const T &in);
00111
00112   // Edge operations
00113   bool addEdge(const T &sourc, const T &dest, double w, std::string route);
00114   bool removeEdge(const T &sourc, const T &dest);
00115
00116   // Graph Information
00117   vector<Vertex<T> *> getVertexSet() const;
00118
00119   // Traversal Operations
00120   vector<T> dfs() const;
00121   vector<T> dfs(const T &source) const;
00122   vector<T> bfs(const T &source) const;
00123
00124   // Topological Sort
00125   vector<T> topsort() const;
00126
00127   // Graph Properties
00128   bool isDAG() const;
00129 };
00130
00131 /****************** Provided constructors and functions ********************/
00132
00138 template <class T>
00139 Vertex<T>::Vertex(T in) : info(in) {}
00140
00148 template <class T>
00149 Edge<T>::Edge(Vertex<T> *d, double w, std::string r)
00150     : dest(d), weight(w), route(r) {}
00151
00157 template <class T>
00158 int Graph<T>::getNumVertex() const
00159 {
00160   return vertexSet.size();
00161 }
00162
00163 // Get vector of vertices in the graph
00169 template <class T>
00170 vector<Vertex<T> *> Graph<T>::getVertexSet() const
00171 {
00172   return vertexSet;
00173 }
00174
00180 template <class T>
00181 T Vertex<T>::getInfo() const { return info; }
00182
00189 template <class T>
00190 void Vertex<T>::setInfo(T in) { Vertex::info = in; }
```

```
00191
00197 template <class T>
00198 bool Vertex<T>::isProcessing() const { return processing; }
00199
00206 template <class T>
00207 void Vertex<T>::setProcessing(bool p)
00208 {
00209   Vertex::processing = p;
00210 }
00211
00217 template <class T>
00218 Vertex<T> *Edge<T>::getDest() const { return dest; }
00219
00226 template <class T>
00227 void Edge<T>::setDest(Vertex<T> *d) { Edge::dest = d; }
00228
00234 template <class T>
00235 double Edge<T>::getWeight() const { return weight; }
00236
00243 template <class T>
00244 void Edge<T>::setWeight(double weight)
00245 {
00246   Edge::weight = weight;
00247 }
00248
00254 template <class T>
00255 std::string Edge<T>::getRoute() const { return route; }
00256
00263 template <class T>
00264 Vertex<T> *Graph<T>::findVertex(const T &in) const
00265 {
00266   for (auto v : vertexSet)
00267     if (v->info == in)
00268       return v;
00269   return NULL;
00270 }
00271
00277 template <class T>
00278 bool Vertex<T>::isVisited() const { return visited; }
00279
00285 template <class T>
00286 int Vertex<T>::getIndegree() const { return indegree; }
00287
00294 template <class T>
00295 void Vertex<T>::setIndegree(int indegree)
00296 {
00297   Vertex::indegree = indegree;
00298 }
00299
00305 template <class T>
00306 int Vertex<T>::getNum() const { return num; }
00307
00314 template <class T>
00315 void Vertex<T>::setNum(int num) { Vertex::num = num; }
00316
00322 template <class T>
00323 int Vertex<T>::getLow() const { return low; }
00324
00331 template <class T>
00332 void Vertex<T>::setLow(int low) { Vertex::low = low; }
00333
00340 template <class T>
00341 void Vertex<T>::setVisited(bool v) { Vertex::visited = v; }
00342
00348 template <class T>
00349 const vector<Edge<T» &Vertex<T>::getAdj() const
00350 {
00351   return adj;
00352 }
00353
00359 template <class T>
00360 void Vertex<T>::setAdj(const vector<Edge<T>> &adj)
00361 {
00362   Vertex::adj = adj;
00363 }
00364
00371 template <class T>
00372 bool Graph<T>::addVertex(const T &in)
00373 {
00374   if (findVertex(in) != NULL)
00375     return false;
00376   vertexSet.push_back(new Vertex<T>(in));
00377   return true;
00378 }
00379
00389 template <class T>
00390 bool Graph<T>::addEdge(const T &sourc, const T &dest, double w,
```

```
00391                         std::string route)
00392 {
00393   auto v1 = findVertex(sourc);
00394   auto v2 = findVertex(dest);
00395   if (v1 == NULL || v2 == NULL)
00396     return false;
00397   v1->addEdge(v2, w, route);
00398   return true;
00399 }
00400
00409 template <class T>
00410 void Vertex<T>::addEdge(Vertex<T> *d, double w, std::string route)
00411 {
00412   adj.push_back(Edge<T>(d, w, route));
00413 }
00414
00422 template <class T>
00423 bool Graph<T>::removeEdge(const T &sourc, const T &dest)
00424 {
00425   auto v1 = findVertex(sourc);
00426   auto v2 = findVertex(dest);
00427   if (v1 == NULL || v2 == NULL)
00428     return false;
00429   return v1->removeEdgeTo(v2);
00430 }
00431
00438 template <class T>
00439 bool Vertex<T>::removeEdgeTo(Vertex<T> *d)
00440 {
00441   for (auto it = adj.begin(); it != adj.end(); it++)
00442     if (it->dest == d)
00443     {
00444       adj.erase(it);
00445       return true;
00446     }
00447   return false;
00448 }
00449
00456 template <class T>
00457 bool Graph<T>::removeVertex(const T &in)
00458 {
00459   for (auto it = vertexSet.begin(); it != vertexSet.end(); it++)
00460     if ((*it)->info == in)
00461     {
00462       auto v = *it;
00463       vertexSet.erase(it);
00464       for (auto u : vertexSet)
00465         u->removeEdgeTo(v);
00466       delete v;
00467       return true;
00468     }
00469   return false;
00470 }
00471
00477 template <class T>
00478 vector<T> Graph<T>::dfs() const
00479 {
00480   vector<T> res;
00481   for (auto v : vertexSet)
00482     v->visited = false;
00483   for (auto v : vertexSet)
00484     if (!v->visited)
00485       dfsVisit(v, res);
00486   return res;
00487 }
00488
00496 template <class T>
00497 void Graph<T>::dfsVisit(Vertex<T> *v, vector<T> &res) const
00498 {
00499   v->visited = true;
00500   res.push_back(v->info);
00501   for (auto &e : v->adj)
00502   {
00503     auto w = e.dest;
00504     if (!w->visited)
00505       dfsVisit(w, res);
00506   }
00507 }
00508
00515 template <class T>
00516 vector<T> Graph<T>::dfs(const T &source) const
00517 {
00518   vector<T> res;
00519   auto s = findVertex(source);
00520   if (s == nullptr)
00521     return res;
00522
```

```
00523   for (auto v : vertexSet)
00524     v->visited = false;
00525
00526   dfsVisit(s, res);
00527   return res;
00528 }
00529
00536 template <class T>
00537 vector<T> Graph<T>::bfs(const T &source) const
00538 {
00539   vector<T> res;
00540   auto s = findVertex(source);
00541   if (s == NULL)
00542     return res;
00543   queue<Vertex<T> *> q;
00544   for (auto v : vertexSet)
00545     v->visited = false;
00546   q.push(s);
00547   s->visited = true;
00548   while (!q.empty())
00549   {
00550     auto v = q.front();
00551     q.pop();
00552     res.push_back(v->info);
00553     for (auto &e : v->adj)
00554     {
00555       auto w = e.dest;
00556       if (!w->visited)
00557       {
00558         q.push(w);
00559         w->visited = true;
00560       }
00561     }
00562   }
00563   return res;
00564 }
00565
00571 template <class T>
00572 bool Graph<T>::isDAG() const
00573 {
00574   for (auto v : vertexSet)
00575   {
00576     v->visited = false;
00577     v->processing = false;
00578   }
00579   for (auto v : vertexSet)
00580     if (!v->visited)
00581       if (!dfsIsDAG(v))
00582         return false;
00583   return true;
00584 }
00585
00592 template <class T>
00593 bool Graph<T>::dfsIsDAG(Vertex<T> *v) const
00594 {
00595   v->visited = true;
00596   v->processing = true;
00597   for (auto &e : v->adj)
00598   {
00599     auto w = e.dest;
00600     if (w->processing)
00601       return false;
00602     if (!w->visited)
00603       if (!dfsIsDAG(w))
00604         return false;
00605   }
00606   v->processing = false;
00607   return true;
00608 }
00609
00615 template <class T>
00616 vector<T> Graph<T>::topsort() const
00617 {
00618   vector<T> res;
00619
00620   for (auto v : vertexSet)
00621   {
00622     v->indegree = 0;
00623   }
00624
00625   for (auto v : vertexSet)
00626   {
00627     for (auto &e : v->adj)
00628     {
00629       e.dest->indegree += 1;
00630     }
00631   }
```

```
00632
00633   std::queue<Vertex<T> *> q;
00634
00635   for (auto v : vertexSet)
00636   {
00637     if (v->indegree == 0)
00638     {
00639       q.push(v);
00640     }
00641   }
00642
00643   while (!q.empty())
00644   {
00645     auto u = q.front();
00646     res.push_back(u->info);
00647     q.pop();
00648
00649     for (auto &e : u->adj)
00650     {
00651       auto w = e.dest;
00652       w->indegree -= 1;
00653       if (w->indegree == 0)
00654       {
00655         q.push(w);
00656       }
00657     }
00658   }
00659   return res;
00660 }
00661
00662 #endif /* GRAPH_H_ */
```

4.11 src/database/dbairport.cpp File Reference

```
#include "dbairport.h"
```

Functions

- int quantityAirports (Graph< Airport > airports)

  Function to get quantity of airports in the graph Complexity: O(1)
- int quantityAirportsCountry (Graph< Airport > airports, std::string country)

  Function to get quantity of airports in the graph by country Complexity: O(n), where n is the number of airports in the graph.
- int quantityAirportsCity (Graph< Airport > airports, std::string city)

  Function to get quantity of airports in the graph by city Complexity: O(n), where n is the number of airports in the graph.
- int quantityCitiesCountry (Graph< Airport > airports, std::string country)

  Function to count the number of unique cities in the graph for a given country Complexity: O(n), where n is the number of airports in the graph.
- int quantityAirlinesCountry (unordered_map< string, Airline > airlines, std::string country)

  Function to calculate the quantity of airlines for a given country Complexity: O(n), where n is the number of airlines in the graph.
- int quantityFlights (Graph< Airport > airports)

  Function to calculate the quatity of flights in the graph Complexity: O(n), where n is the number of airports in the graph.
- int quantityFlightsAirport (Graph< Airport > airports, std::string airport)

  Function to calculate the quatity of flights in a given airport Complexity: O(n), where n is the number of airports in the graph.
- int quantityFlightsCountry (Graph< Airport > airports, std::string country)

  Function to calculate the quatity of flights in a given country Complexity: O(n), where n is the number of airports in the graph.
- int quantityFlightsCity (Graph< Airport > airports, std::string city)

Function to calculate the quatity of flights in a given city Complexity: O(n), where n is the number of airports in the graph.

- int quantityFlightsAirline (Graph< Airport > airports, std::string airline)

  Function to calculate the quatity of flights in a given airline Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.
- int quantityDestinationsAirport (Graph< Airport > airports, std::string airport)

  Function to calculate the number of different countries it is possible to go directly to from an airport Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.
- int quantityDestinationLimitedStop (Graph< Airport > airports, std::string airport, int stop)

  Function to calculate the number of different countries it is possible to go to from an airport with a given number of stops Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.
- int quantityDestinationMax (Graph< Airport > airports)

  Function that calculates the maximum possible trip, with the greatest number of stops and print the starting airport and the destination airport Complexity: O(n ∗ (n + e)), where n is the number of airports in the graph and e is the number of flights.
- void dfsMax (Vertex< Airport > ∗v, std::vector< std::string > &path, std::vector< std::string > &maxPath)

  Function that performs a depth-first search and always updates the longest path.
- bool comparatorPath (const vector< Flight > a, const vector< Flight > b)

  Function to sort the vector with the best flights in increasing order by the number of stops.
- std::pair< int, int > quantityFlights (Graph< Airport > airports, std::string code)

  Function to get quantity of flights in the graph and unique airlines for an Airport Complexity: O(n), where n is the number of flights in the graph.
- std::vector< std::string > dfsVisit (Vertex< Airport > ∗v, std::vector< std::string > &res)

  Search (depth-first) to visit vertices and collect connected countries.
- std::vector< std::string > dfsVisit (Vertex< Airport > ∗v, std::vector< std::string > &res, int stop)

  Search (depth-first) to visit vertices and collect connected countries withing a number of stops.
- void resetVisited (Graph< Airport > &airports)

  Function to reset the visited status of all vertices in the graph Complexity: O(n), where n is the number of airports in the graph.
- void calculateIndegree (Graph< Airport > &airports)

  Function to calculate the in-degreee of each vertex in the graph Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void rankingAirports (Graph< Airport > airports, int arg)

  Function to rank airport based of the sum of in-degrees and out-degrees Complexity: O(n ∗ log(n) + n ∗ e), where n is the number of airports and e is the number of flights in the graph (assuming sorting has a time complexity of O(n ∗ log(n)))
- void findBestFlights (Graph< Airport > &airports, string src, string dest, vector< string > &airplanes)

  Function to find the best flights from a source airport to a destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findBestFlights (Graph< Airport > &airports, string country, string city, string airport, int type, vector< string > &airplanes)

  Finds the best flights between a city and an aiport (vice-verse) This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void showPath (Graph< Airport > &airports, vector< Vertex< Airport > ∗ > source, vector< Vertex< Airport > ∗ > dest, vector< vector< Flight > > paths, vector< string > &airplanes)

  Function to show a multi-path between multiple source and destination airports Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void showPath (vector< vector< Flight > > paths)

  Function to show a set of paths Complexity: O(n), where n is the total number of flights in all paths.
- vector< Vertex< Airport > ∗ > findAirports (Graph< Airport > &airports, string country, string city)

  Function to find all airports in a city Complexity: O(n), where n is the number of airports in the graph.
- void findBestFlights (Graph< Airport > &airports, string countrySrc, string citySrc, string countryDest, string cityDest, vector< string > &airplanes)

Function to find best flights between two cities This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

- double toRadians (const double degree)

  Function to convert Degrees in radians Complexity: O(1)
- double distanceEarth (double latOrigin, double longOrigin, double latDest, double longDest)

  Function to calculate the distance between two points on the Earth Complexity: O(1)
- vector< Vertex< Airport > * > findAirports (Graph< Airport > &airports, double lat, double lon, int distMax)

  Function to find all airports around a point withing a distance Complexity: O(n), where n is the number of airports in the graph.
- void findBestFlights (Graph< Airport > &airports, double latOrigin, double longOrigin, double latDest, double longDest, int distMax, vector< string > &airplanes)

  Function to find the best flights between two points on Earth Point to Point This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findBestFlights (Graph< Airport > &airports, string airport, double lat, double lon, int distMax, int type, vector< string > &airplanes)

  Function to find best flights between an airport and a point on Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.*
- void findBestFlights (Graph< Airport > &airports, string country, string city, double lat, double lon, int distMax, int type, vector< string > &airplanes)

  Function to find best flights between a city and a point on the Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.*
- void getPath (string current, vector< Flight > &path, unordered_map< string, vector< Flight > > &prev, vector< vector< Flight > > &paths, string startCode, string airline)

  Function to find a path from the current airport to start airport Complexity: O(n), where n is the number of airports in the graph.
- vector< vector< Flight > > bfsPath (Vertex< Airport > *v, string &tgt, vector< string > &airplanes)

  Function to perfom breadth-first search to find paths between two airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- int connectedComponents (Graph< Airport > &airports)

  Function that uses a depth-first search to find the number of connected components in a graph. Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void dfsConnectedComponents (Graph< Airport > &airports, Vertex< Airport > *v)

  Function DFS starting from a vertex to find connected components Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findArticulationPoints (Graph< Airport > &airports)

  Function to find and show articulation points in the graph. Points (airports) that, if removed, make certain regions of the map inaccessible Complexity: O(n^2), where n is the number of airports in the graph.
- void dfsArtc (Vertex< Airport > *v, Vertex< Airport > *w)

  Function DFS modified to find articulation points.

4.11.1 Function Documentation

4.11.1.1 bfsPath()

```
vector< vector< Flight > > bfsPath (
            Vertex< Airport > * v,
            string & tgt,
            vector< string > & airplanes )
```

Function to perfom breadth-first search to find paths between two airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

v	The vertex to start the search
tgt	The target airport.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

vector<vector<Flight>> A vector of vectors of flights representing the paths.

Definition at line 964 of file dbairport.cpp.

```
00966 {
00967     vector<vector<Flight>> paths;
00968     unordered_map<string, vector<Flight>> prev;
00969     unordered_map<string, int> dist;
00970     queue<Vertex<Airport> *> q;
00971
00972     q.push(v);
00973     v->setVisited(true);
00974     dist[v->getInfo().getCode()] = 0;
00975
00976     while (!q.empty())
00977     {
00978         auto vertex = q.front();
00979         q.pop();
00980         auto adjs = vertex->getAdj();
00981
00982         for (auto &e : adjs)
00983         {
00984             auto w = e.getDest();
00985             if (!w->isVisited() && (find(airplanes.begin(), airplanes.end(),
00986                                          e.getRoute()) != airplanes.end() ||
00987                                     airplanes.size() == 0))
00988             {
00989                 q.push(w);
00990                 w->setVisited(true);
00991                 prev[w->getInfo().getCode()].push_back(
00992                     {vertex->getInfo().getCode(), e.getRoute()});
00993                 dist[w->getInfo().getCode()] = dist[vertex->getInfo().getCode()] + 1;
00994             }
00995             else if (dist[w->getInfo().getCode()] ==
00996                          dist[vertex->getInfo().getCode()] + 1 &&
00997                      (find(airplanes.begin(), airplanes.end(), e.getRoute()) !=
00998                           airplanes.end() ||
00999                       airplanes.size() == 0))
01000             {
01001                 prev[w->getInfo().getCode()].push_back(
01002                     {vertex->getInfo().getCode(), e.getRoute()});
01003             }
01004         }
01005     }
01006
01007     vector<Flight> path;
01008     getPath(tgt, path, prev, paths, v->getInfo().getCode(), "");
01009
01010     return paths;
01011 }
```

4.11.1.2 calculateIndegree()

```
void calculateIndegree (
            Graph< Airport > & airports )
```

Function to calculate the in-degreee of each vertex in the graph Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph of airports.

Returns

void

Definition at line 436 of file dbairport.cpp.

```
00437 {
00438     for (auto v : airports.getVertexSet())
00439     {
00440         v->setIndegree(0);
00441     }
00442     for (auto v : airports.getVertexSet())
00443     {
00444         for (auto e : v->getAdj())
00445         {
00446             e.getDest()->setIndegree(e.getDest()->getIndegree() + 1);
00447         }
00448     }
00449 }
```

4.11.1.3 comparatorPath()

```
bool comparatorPath (
            const vector< Flight > a,
            const vector< Flight > b )
```

Function to sort the vector with the best flights in increasing order by the number of stops.

Parameters

a	The first vector.
b	The second vector.

Returns

bool True if the first vector is smaller than the second vector.

Definition at line 326 of file dbairport.cpp.

```
00327 {
00328     return a.size() < b.size();
00329 }
```

4.11.1.4 connectedComponents()

```
int connectedComponents (
            Graph< Airport > & airports )
```

Function that uses a depth-first search to find the number of connected components in a graph. Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.

Returns

int The number of connected components.

Definition at line 1019 of file dbairport.cpp.

```
01020 {
01021     int component = 0;
01022     resetVisited(airports);
01023     Graph<Airport> g = airports;
01024
01025     for (auto v : g.getVertexSet())
01026     {
01027         if (!v->isVisited())
01028         {
01029             dfsConnectedComponents(g, v);
01030             component++;
01031         }
01032     }
01033     // std::cout « "Number of connected components: " « component «
01034     // std::endl;
01035     return component;
01036 }
```

4.11.1.5 dfsArtc()

```
void dfsArtc (
            Vertex< Airport > * v,
            Vertex< Airport > * w )
```

Function DFS modified to find articulation points.

Parameters

v	The vertex to start the search
w	The vertex to be visited

Returns

void

Definition at line 1109 of file dbairport.cpp.

```
01110 {
01111     w->setVisited(true);
01112     auto adjs = w->getAdj();
01113
01114     for (auto &e : adjs)
01115     {
01116         auto t = e.getDest();
01117         if (!t->isVisited() && t->getInfo().getCode() != v->getInfo().getCode())
01118         {
01119             dfsArtc(v, t);
01120         }
01121     }
01122 }
```

4.11.1.6 dfsConnectedComponents()

```
void dfsConnectedComponents (
            Graph< Airport > & airports,
            Vertex< Airport > * v )
```

Function DFS starting from a vertex to find connected components Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
v	The vertex to start the search

Returns

void

Definition at line 1045 of file dbairport.cpp.

```
01046 {
01047     v->setVisited(true);
01048     auto adjs = v->getAdj();
01049
01050     for (auto &e : adjs)
01051     {
01052         auto w = e.getDest();
01053         if (!w->isVisited())
01054         {
01055             dfsConnectedComponents(airports, w);
01056         }
01057     }
01058 }
```

4.11.1.7 dfsMax()

```
void dfsMax (
            Vertex< Airport > * v,
            std::vector< std::string > & path,
            std::vector< std::string > & maxPath )
```

Function that performs a depth-first search and always updates the longest path.

Parameters

v	The vertex to start the search
path	The current path
maxPath	The longest path

Returns

void

Definition at line 293 of file dbairport.cpp.

```
00294 {
00295     v->setVisited(true);
00296     path.push_back(v->getInfo().getCode());
00297     auto adjs = v->getAdj();
00298
00299     if (path.size() > maxPath.size())
00300     {
00301         maxPath = path;
00302     }
00303
00304     for (auto &e : adjs)
00305     {
00306         auto w = e.getDest();
00307
00308         if (!w->isVisited())
00309         {
00310             dfsMax(w, path, maxPath);
00311         }
00312     }
00313     path.pop_back();
00314 }
```

4.11.1.8 dfsVisit() [1/2]

```
std::vector< std::string > dfsVisit (
            Vertex< Airport > * v,
            std::vector< std::string > & res )
```

Search (depth-first) to visit vertices and collect connected countries.

Parameters

airports	The graph of airports.
v	The vertex to start the search

Returns

vector<string> The vector with the countries.

Definition at line 361 of file dbairport.cpp.

```
00363 {
00364     v->setVisited(true);
00365     res.push_back(v->getInfo().getCountry());
00366     auto adjs = v->getAdj();
00367 
00368     for (auto &e : adjs)
00369     {
00370         auto w = e.getDest();
00371 
00372         if (!w->isVisited())
00373         {
00374             dfsVisit(w, res);
00375         }
00376     }
00377 
00378     return res;
00379 }
```

4.11.1.9 dfsVisit() [2/2]

```
std::vector< std::string > dfsVisit (
            Vertex< Airport > * v,
            std::vector< std::string > & res,
            int stop )
```

Search (depth-first) to visit vertices and collect connected countries withing a number of stops.

Parameters

airports	The graph of airports.
v	The vertex to start the search
stop	The number of stops.

Returns

vector<string> The vector with the countries.

Definition at line 389 of file dbairport.cpp.

```
00391 {
00392     if (stop == 0)
00393         return res;
00394 
00395     // std::cout << "Saindo de " << v->getInfo().getCode() << std::endl;
00396 
00397     v->setVisited(true);
00398     res.push_back(v->getInfo().getCountry());
00399     auto adjs = v->getAdj();
00400 
00401     for (auto &e : adjs)
```

```
00402     {
00403         auto w = e.getDest();
00404 
00405         if (!w->isVisited())
00406         {
00407             // std::cout « "\t Eu vou para: " « w->getInfo().getCode() «
00408             // std::endl;
00409             dfsVisit(w, res, (stop - 1));
00410         }
00411     }
00412     return res;
00413 }
```

4.11.1.10 distanceEarth()

```
double distanceEarth (
            double latOrigin,
            double longOrigin,
            double latDest,
            double longDest )
```

Function to calculate the distance between two points on the Earth Complexity: O(1)

Parameters

latOrigin	The latitude of the origin point.
longOrigin	The longitude of the origin point.
latDest	The latitude of the destination point.
longDest	The longitude of the destination point.

Returns

double The distance between the two points.

Definition at line 735 of file dbairport.cpp.

```
00737 {
00738     latOrigin = toRadians(latOrigin);
00739     longOrigin = toRadians(longOrigin);
00740     latDest = toRadians(latDest);
00741     longDest = toRadians(longDest);
00742 
00743     double dlong = longDest - longOrigin;
00744     double dlat = latDest - latOrigin;
00745 
00746     double ans = pow(sin(dlat / 2), 2) +
00747                  cos(latOrigin) * cos(latDest) * pow(sin(dlong / 2), 2);
00748 
00749     ans = 2 * asin(sqrt(ans));
00750 
00751     double R = 6371;
00752 
00753     ans = ans * R;
00754 
00755     return ans;
00756 }
```

4.11.1.11 findAirports() [1/2]

```
vector< Vertex< Airport > * > findAirports (
            Graph< Airport > & airports,
            double lat,
```

```
        double lon,
        int distMax )
```

Function to find all airports around a point withing a distance Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph representing airports and available flights.
lat	The latitude of the point.
lon	The longitude of the point.
distMax	The maximum distance.

Returns

vector<Vertex<Airport>*> A vector of airports.

Definition at line 770 of file dbairport.cpp.

```
00772 {
00773     vector<Vertex<Airport> *> vec;
00774     for (auto v : airports.getVertexSet())
00775     {
00776         double lat2 = v->getInfo().getLatitude();
00777         double lon2 = v->getInfo().getLongitude();
00778 
00779         double dist = distanceEarth(lat, lon, lat2, lon2);
00780 
00781         if (dist <= distMax)
00782         {
00783             vec.push_back(v);
00784         }
00785     }
00786     return vec;
00787 }
```

4.11.1.12 findAirports() [2/2]

```
vector< Vertex< Airport > * > findAirports (
            Graph< Airport > & airports,
            string country,
            string city )
```

Function to find all airports in a city Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph representing airports and available flights.
country	The country of the airport.
city	The city of the airport.

Returns

vector<Vertex<Airport>*> A vector of airports.

Definition at line 671 of file dbairport.cpp.

```
00673 {
00674     vector<Vertex<Airport> *> vec;
00675     for (auto v : airports.getVertexSet())
00676     {
00677         if (v->getInfo().getCountry() == country &&
00678             v->getInfo().getCity() == city)
00679         {
00680             vec.push_back(v);
00681         }
00682     }
00683     return vec;
00684 }
```

4.11.1.13 findArticulationPoints()

```
void findArticulationPoints (
            Graph< Airport > & airports )
```

Function to find and show articulation points in the graph. Points (airports) that, if removed, make certain regions of the map inaccessible Complexity: O(n^2), where n is the number of airports in the graph.

Parameters

airports	The graph representing airports and available flights.

Returns

void

Definition at line 1066 of file dbairport.cpp.

```
01067 {
01068     int i = 0;
01069     int total = 0;
01070     int connected = connectedComponents(airports);
01071     for (auto v : airports.getVertexSet())
01072     {
01073         // std::cout « " Analisando o vertice: " « v->getInfo().getCode() «
01074         // std::endl;
01075         resetVisited(airports);
01076         int component = 0;
01077         for (auto w : airports.getVertexSet())
01078         {
01079             // std::cout « "\tAnalisando o vertice: " « w->getInfo().getCode()«
01080             // std::endl;
01081             if (!w->isVisited() && w->getInfo().getCode() != v->getInfo().getCode())
01082             {
01083                 // std::cout « "\t\tChamando o DFS component++" « std::endl;
01084                 dfsArtc(v, w);
01085                 component++;
01086             }
01087         }
01088         if (component > connected)
01089         {
01090             std::cout « v->getInfo().getCode() « " ";
01091             i++;
01092             total++;
01093             if (i == 10)
01094             {
01095                 std::cout « std::endl;
01096                 i = 0;
01097             }
01098         }
01099     }
01100     std::cout « "\nTotal: " « total « std::endl;
01101 }
```

4.11.1.14 findBestFlights() [1/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            double latOrigin,
            double longOrigin,
            double latDest,
            double longDest,
            int distMax,
            vector< string > & airplanes )
```

Function to find the best flights between two points on Earth Point to Point This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
latOrigin	The latitude of the origin point.
longOrigin	The longitude of the origin point.
latDest	The latitude of the destination point.
longDest	The longitude of the destination point.
distMax	The maximum distance.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 802 of file dbairport.cpp.

```
00805 {
00806     vector<Vertex<Airport> *> src =
00807         findAirports(airports, latOrigin, longOrigin, distMax);
00808     vector<Vertex<Airport> *> dest =
00809         findAirports(airports, latDest, longDest, distMax);
00810     vector<vector<Flight>> paths;
00811
00812     showPath(airports, src, dest, paths, airplanes);
00813 }
```

4.11.1.15 findBestFlights() [2/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string airport,
            double lat,
            double lon,
            int distMax,
            int type,
            vector< string > & airplanes )
```

Function to find best flights between an airport and a point on Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p∗(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.

Parameters

airports	The graph representing airports and available flights.
airport	The airport.
lat	The latitude of the point.
lon	The longitude of the point.
distMax	The maximum distance.
type	The type of search (0: Point to Airport, 1: Airport to Point).
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 828 of file dbairport.cpp.

```
00831 {
00832     vector<Vertex<Airport> *> vec;
00833     vector<vector<Flight>> paths;
00834     vec = findAirports(airports, lat, lon, distMax);
00835
00836     if (type == 0)
00837     {
00838         for (auto v : vec)
00839         {
00840             std::cout << "Source: " << v->getInfo().getCode() << std::endl;
00841             resetVisited(airports);
00842             paths = bfsPath(v, airport, airplanes);
00843             showPath(paths);
00844         }
00845     }
00846     else if (type == 1)
00847     {
00848         for (auto v : vec)
00849         {
00850             std::cout << "Source: " << airport << std::endl;
00851             std::string tgt = v->getInfo().getCode();
00852             resetVisited(airports);
00853             paths = bfsPath(airports.findVertex(Airport(airport)), tgt, airplanes);
00854             showPath(paths);
00855         }
00856     }
00857 }
```

4.11.1.16 findBestFlights() [3/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string country,
            string city,
            double lat,
            double lon,
            int distMax,
            int type,
            vector< string > & airplanes )
```

Function to find best flights between a city and a point on the Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p∗(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.

Parameters

airports	The graph representing airports and available flights.
country	The country of the source airport.
city	The city of the source airport.
lat	The latitude of the point.
lon	The longitude of the point.
distMax	The maximum distance.
type	The type of search (0: City to Point, 1: Point to City).
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 878 of file dbairport.cpp.

```
00881 {
```

```
00882     vector<Vertex<Airport> *> vec_point;
00883     vector<Vertex<Airport> *> vec_city;
00884     vector<vector<Flight>> paths;
00885 
00886     vec_point = findAirports(airports, lat, lon, distMax);
00887     vec_city = findAirports(airports, country, city);
00888 
00889     if (type == 0)
00890     {
00891         for (auto c : vec_city)
00892         {
00893             std::cout « "Source: " « c->getInfo().getCode() « std::endl;
00894             for (auto p : vec_point)
00895             {
00896                 std::cout « "Destination: " « p->getInfo().getCode() « std::endl;
00897                 resetVisited(airports);
00898                 std::string tgt = p->getInfo().getCode();
00899                 paths = bfsPath(c, tgt, airplanes);
00900                 showPath(paths);
00901             }
00902         }
00903     }
00904     else if (type == 1)
00905     {
00906         for (auto p : vec_point)
00907         {
00908             std::cout « "Source: " « p->getInfo().getCode() « std::endl;
00909             for (auto c : vec_city)
00910             {
00911                 std::cout « "Destination: " « c->getInfo().getCode() « std::endl;
00912                 resetVisited(airports);
00913                 std::string tgt = c->getInfo().getCode();
00914                 paths = bfsPath(p, tgt, airplanes);
00915                 showPath(paths);
00916             }
00917         }
00918     }
00919 }
```

4.11.1.17 findBestFlights() [4/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string country,
            string city,
            string airport,
            int type,
            vector< string > & airplanes )
```

Finds the best flights between a city and an aiport (vice-verse) This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
country	The country of the source airport.
city	The city of the source airport.
airport	The destination airport.
type	The type of search (0: City to Airport, 1: Airport to City).
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 522 of file dbairport.cpp.

```
00524 {
00525     vector<Vertex<Airport> *> vec;
00526     vector<vector<Flight>> paths;
00527     vec = findAirports(airports, country, city);
00528 
00529     if (type == 0)
00530     {
00531         for (auto v : vec)
00532         {
00533             // std::cout « "Source: " « v->getInfo().getCode() « std::endl;
00534             resetVisited(airports);
00535             paths = bfsPath(v, airport, airplanes);
00536             showPath(paths);
00537         }
00538     }
00539     else if (type == 1)
00540     {
00541         for (auto v : vec)
00542         {
00543             // std::cout « "Source: " « airport « std::endl;
00544             std::string tgt = v->getInfo().getCode();
00545             resetVisited(airports);
00546             paths = bfsPath(airports.findVertex(Airport(airport)), tgt, airplanes);
00547             showPath(paths);
00548         }
00549     }
00550 }
```

4.11.1.18 findBestFlights() [5/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string countrySrc,
            string citySrc,
            string countryDest,
            string cityDest,
            vector< string > & airplanes )
```

Function to find best flights between two cities This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
countrySrc	The country of the source airport.
citySrc	The city of the source airport.
countryDest	The country of the destination airport.
cityDest	The city of the destination airport.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 698 of file dbairport.cpp.

```
00701 {
00702     vector<Vertex<Airport> *> src;
00703     vector<Vertex<Airport> *> dest;
00704     vector<vector<Flight>> paths;
00705 
00706     src = findAirports(airports, countrySrc, citySrc);
00707     dest = findAirports(airports, countryDest, cityDest);
00708 
00709     showPath(airports, src, dest, paths, airplanes);
00710 }
```

4.11.1.19 findBestFlights() [6/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string src,
            string dest,
            vector< string > & airplanes )
```

Function to find the best flights from a source airport to a destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
src	The source airport.
dest	The destination airport.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 494 of file dbairport.cpp.

```
00496 {
00497     resetVisited(airports);
00498
00499     auto s = airports.findVertex(Airport(src));
00500     auto d = airports.findVertex(Airport(dest));
00501
00502     if (s == nullptr || d == nullptr)
00503         return;
00504
00505     vector<vector<Flight>> paths;
00506     paths = bfsPath(s, dest, airplanes);
00507     showPath(paths);
00508 }
```

4.11.1.20 getPath()

```
void getPath (
            string current,
            vector< Flight > & path,
            unordered_map< string, vector< Flight > > & prev,
            vector< vector< Flight > > & paths,
            string startCode,
            string airline )
```

Function to find a path from the current airport to start airport Complexity: O(n), where n is the number of airports in the graph.

Parameters

current	The current airport.
path	The current path.
prev	A map with the previous airports.
paths	A vector of vectors of flights representing the paths.
startCode	The code of the start airport.
airline	The airline of the flight.

Returns

void

Definition at line 932 of file dbairport.cpp.

```
00935 {
00936     path.push_back({current, airline});
00937
00938     if (current == startCode)
00939     {
00940         vector<Flight> validPath = path;
00941         reverse(validPath.begin(), validPath.end());
00942         paths.push_back(validPath);
00943     }
00944     else
00945     {
00946         for (auto &prevVertex : prev[current])
00947         {
00948             getPath(prevVertex.code, path, prev, paths, startCode,
00949                     prevVertex.airline);
00950         }
00951     }
00952
00953     path.pop_back();
00954 }
```

4.11.1.21 quantityAirlinesCountry()

```
int quantityAirlinesCountry (
            unordered_map< string, Airline > airlines,
            std::string country )
```

Function to calculate the quantity of airlines for a given country Complexity: O(n), where n is the number of airlines in the graph.

Parameters

airlines	The airlines hashtable.
country	The country to be searched.

Returns

int The quantity of airlines in the country.

Definition at line 90 of file dbairport.cpp.

```
00091 {
00092     int count = 0;
00093     for (auto v : airlines)
00094     {
00095         if (v.second.getCountry() == country)
00096         {
00097             count++;
00098         }
00099     }
00100     return count;
00101 }
```

4.11.1.22 quantityAirports()

```
int quantityAirports (
            Graph< Airport > airports )
```

Function to get quantity of airports in the graph Complexity: O(1)

Parameters

airports	The graph of airports.

Returns

int The quantity of airports in the graph.

Definition at line 11 of file dbairport.cpp.

```
00012 {
00013     return airports.getNumVertex();
00014 }
```

4.11.1.23 quantityAirportsCity()

```
int quantityAirportsCity (
            Graph< Airport > airports,
            std::string city )
```

Function to get quantity of airports in the graph by city Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
city	The city to be searched.

Returns

int The quantity of airports in the city.

Definition at line 44 of file dbairport.cpp.

```
00045 {
00046     int count = 0;
00047     for (auto v : airports.getVertexSet())
00048     {
00049         Airport a = v->getInfo();
00050         if (a.getCity() == city)
00051         {
00052             count++;
00053         }
00054     }
00055     return count;
00056 }
```

4.11.1.24 quantityAirportsCountry()

```
int quantityAirportsCountry (
            Graph< Airport > airports,
            std::string country )
```

Function to get quantity of airports in the graph by country Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
country	The country to be searched.

Returns

int The quantity of airports in the country.

Definition at line 23 of file dbairport.cpp.

```
00024 {
00025     int count = 0;
00026     for (auto v : airports.getVertexSet())
00027     {
00028         Airport a = v->getInfo();
00029         if (a.getCountry() == country)
00030         {
00031             count++;
00032         }
00033     }
00034     return count;
00035 }
```

4.11.1.25 quantityCitiesCountry()

```
int quantityCitiesCountry (
            Graph< Airport > airports,
            std::string country )
```

Function to count the number of unique cities in the graph for a given country Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
country	The country to be searched.

Returns

int The quantity of unique cities in the country.

Definition at line 65 of file dbairport.cpp.

```
00066 {
00067     std::unordered_set<std::string> uniqueCities;
00068     for (auto v : airports.getVertexSet())
00069     {
00070         Airport a = v->getInfo();
00071         if (a.getCountry() == country)
00072         {
00073             std::string city = a.getCity();
00074             uniqueCities.insert(city);
00075         }
00076     }
00077     return uniqueCities.size();
00078 }
```

4.11.1.26 quantityDestinationLimitedStop()

```
int quantityDestinationLimitedStop (
            Graph< Airport > airports,
```

```
            std::string airport,
            int stop )
```

Function to calculate the number of different countries it is possible to go to from an airport with a given number of stops Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.
airport	The airport to be searched.
stop	The number of stops.

Returns

int The quantity of countries.

Definition at line 241 of file dbairport.cpp.

```
00243 {
00244     std::set<std::string> countries;
00245     std::vector<std::string> res;
00246     resetVisited(airports);
00247     auto s = airports.findVertex(Airport(airport));
00248     if (s == nullptr)
00249         return 0;
00250     res = dfsVisit(s, res, (stop + 1));
00251     for (auto c : res)
00252     {
00253         countries.insert(c);
00254     }
00255     return countries.size();
00256 }
```

4.11.1.27 quantityDestinationMax()

```
int quantityDestinationMax (
            Graph< Airport > airports )
```

Function that calculates the maximum possible trip, with the greatest number of stops and print the starting airport and the destination airport Complexity: O(n $*$ (n + e)), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.

Returns

int The number of stops on the longest trip

Definition at line 265 of file dbairport.cpp.

```
00266 {
00267     vector<string> path, maxPath;
00268
00269     for (auto v : airports.getVertexSet())
00270     {
00271         // std::cout << "Consultando... " << v->getInfo().getCode() << std::endl;
00272         resetVisited(airports);
00273         path.clear();
00274         dfsMax(v, path, maxPath);
```

```
00275     }
00276
00277     std::cout « "\nStarting in: " « maxPath[0] « std::endl;
00278     // for (int i = 1; i < maxPath.size() - 1; i++)
00279     // {
00280     //     std::cout « maxPath[i] « " -> ";
00281     // }
00282     std::cout « "\nEnding in: " « maxPath[maxPath.size() - 1] « std::endl;
00283     return maxPath.size() - 1;
00284 }
```

4.11.1.28 quantityDestinationsAirport()

```
int quantityDestinationsAirport (
            Graph< Airport > airports,
            std::string airport )
```

Function to calculate the number of different countries it is possible to go directly to from an airport Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.
airport	The airport to be searched.

Returns

int The quantity of countries.

Definition at line 216 of file dbairport.cpp.

```
00217 {
00218     std::set<std::string> countries;
00219     std::vector<std::string> res;
00220     resetVisited(airports);
00221     auto s = airports.findVertex(Airport(airport));
00222     if (s == nullptr)
00223         return 0;
00224     Graph<Airport> g;
00225     res = dfsVisit(s, res);
00226     for (auto c : res)
00227     {
00228         countries.insert(c);
00229     }
00230     return countries.size();
00231 }
```

4.11.1.29 quantityFlights() [1/2]

```
int quantityFlights (
            Graph< Airport > airports )
```

Function to calculate the quatity of flights in the graph Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.

Returns

int The quantity of flights (edges) in the graph.

Definition at line 111 of file dbairport.cpp.

```
00112 {
00113     int count = 0;
00114     for (auto v : airports.getVertexSet())
00115     {
00116         count += v->getAdj().size();
00117     }
00118     return count;
00119 }
```

4.11.1.30 quantityFlights() [2/2]

```
std::pair< int, int > quantityFlights (
            Graph< Airport > airports,
            std::string code )
```

Function to get quantity of flights in the graph and unique airlines for an Airport Complexity: O(n), where n is the number of flights in the graph.

Parameters

airports	The graph of airports.
code	The airport to be searched.

Returns

pair<int, int> The quantity of flights and unique airlines.

Definition at line 338 of file dbairport.cpp.

```
00339 {
00340     int count = 0;
00341     std::set<std::string> airlines;
00342
00343     auto s = airports.findVertex(Airport(code));
00344     if (s != nullptr)
00345     {
00346         for (auto v : s->getAdj())
00347         {
00348             count++;
00349             airlines.insert(v.getRoute());
00350         }
00351     }
00352     return std::pair<int, int>(count, airlines.size());
00353 }
```

4.11.1.31 quantityFlightsAirline()

```
int quantityFlightsAirline (
            Graph< Airport > airports,
            std::string airline )
```

Function to calculate the quatity of flights in a given airline Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.
airline	The airline to be searched.

Returns

int The quantity of flights (edges) in the airline.

Definition at line 190 of file dbairport.cpp.

```
00191 {
00192     int count = 0;
00193     for (auto v : airports.getVertexSet())
00194     {
00195         for (auto e : v->getAdj())
00196         {
00197             if (e.getRoute() == airline)
00198             {
00199                 count++;
00200             }
00201         }
00202     }
00203
00204     return count;
00205 }
```

4.11.1.32 quantityFlightsAirport()

```
int quantityFlightsAirport (
            Graph< Airport > airports,
            std::string airport )
```

Function to calculate the quatity of flights in a given airport Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
airport	The airport to be searched.

Returns

int The quantity of flights (edges) in the airport.

Definition at line 128 of file dbairport.cpp.

```
00129 {
00130     int count = 0;
00131     for (auto v : airports.getVertexSet())
00132     {
00133         if (v->getInfo().getCode() == airport)
00134         {
00135             count += v->getAdj().size();
00136         }
00137     }
00138     return count;
00139 }
```

4.11.1.33 quantityFlightsCity()

```
int quantityFlightsCity (
            Graph< Airport > airports,
            std::string city )
```

Function to calculate the quatity of flights in a given city Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
city	The city to be searched.

Returns

int The quantity of flights (edges) in the city.

Definition at line 169 of file dbairport.cpp.

```
00170 {
00171     int count = 0;
00172     for (auto v : airports.getVertexSet())
00173     {
00174         if (v->getInfo().getCity() == city)
00175         {
00176             count += v->getAdj().size();
00177         }
00178     }
00179     return count;
00180 }
```

4.11.1.34 quantityFlightsCountry()

```
int quantityFlightsCountry (
            Graph< Airport > airports,
            std::string country )
```

Function to calculate the quatity of flights in a given country Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
country	The country to be searched.

Returns

int The quantity of flights (edges) in the country.

Definition at line 148 of file dbairport.cpp.

```
00149 {
00150     int count = 0;
00151     for (auto v : airports.getVertexSet())
00152     {
00153         if (v->getInfo().getCountry() == country)
00154         {
00155             count += v->getAdj().size();
00156         }
00157     }
00158     return count;
00159 }
```

4.11.1.35 rankingAirports()

```
void rankingAirports (
            Graph< Airport > airports,
            int arg )
```

Function to rank airport based of the sum of in-degrees and out-degrees Complexity: O(n ∗ log(n) + n ∗ e), where n is the number of airports and e is the number of flights in the graph (assuming sorting has a time complexity of O(n ∗ log(n)))

Parameters

airports	The graph of airports.
arg	The number of airports to be shown.

Returns

void

Definition at line 460 of file dbairport.cpp.

```
00461 {
00462     std::vector<Ranking> vec;
00463     calculateIndegree(airports);
00464 
00465     for (auto v : airports.getVertexSet())
00466     {
00467         int total = v->getIndegree() + v->getAdj().size();
00468         Ranking rank = {v->getInfo().getCode(), total};
00469         vec.push_back(rank);
00470     }
00471     std::sort(vec.begin(), vec.end(), comparator);
00472     int i = 0;
00473     for (auto v : vec)
00474     {
00475         if (i < arg)
00476         {
00477             std::cout << "Code: " << v.code << " / ";
00478             std::cout << "Name: " << airportsHash.find(v.code)->second.getName() << " / ";
00479             std::cout << "Total: " << v.count << std::endl;
00480             i++;
00481         }
00482     }
00483 }
```

4.11.1.36 resetVisited()

```
void resetVisited (
            Graph< Airport > & airports )
```

Function to reset the visited status of all vertices in the graph Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.

Returns

void

Definition at line 421 of file dbairport.cpp.

```
00422 {
00423     for (auto v : airports.getVertexSet())
00424     {
00425         v->setVisited(false);
00426     }
00427 }
```

4.11.1.37 showPath() [1/2]

```
void showPath (
            Graph< Airport > & airports,
            vector< Vertex< Airport > * > source,
            vector< Vertex< Airport > * > dest,
            vector< vector< Flight > > paths,
            vector< string > & airplanes )
```

Function to show a multi-path between multiple source and destination airports Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
source	A vector of source airports.
dest	A vector of destination airports.
paths	A vector of vectors of flights representing the paths.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 563 of file dbairport.cpp.

```
00566 {
00567
00568     vector<vector<Flight>> flights;
00569
00570     for (auto s : source)
00571     {
00572         // std::cout « "Source: " « s->getInfo().getCode() « std::endl;
00573         for (auto d : dest)
00574         {
00575             // std::cout « "Destination: " « d->getInfo().getCode() «
00576             // std::endl;
00577             std::string tgt = d->getInfo().getCode();
00578             resetVisited(airports);
00579             flights = bfsPath(s, tgt, airplanes);
00580             paths.insert(paths.end(), flights.begin(), flights.end());
00581         }
00582     }
00583
00584     if (paths.empty())
00585     {
00586         std::cout « "-------------------------------------------------"
00587                   « std::endl;
00588         std::cout « "Sorry, but there is no result with those inputs."
00589                   « std::endl;
00590         return;
00591     }
00592     else
00593     {
00594
00595         std::sort(paths.begin(), paths.end(), comparatorPath);
00596         const int min = paths[0].size();
00597
00598         for (auto &p : paths)
00599         {
00600             if (p.size() > min)
00601             {
00602                 continue;
00603             }
00604             std::cout « "-------------------------------------------------"
00605                       « std::endl;
00606             for (auto &f : p)
00607             {
00608                 if (!f.airline.empty())
00609                 {
00610                     std::cout « f.code « " -(" « f.airline « ")"
```

```
00611                                « "-> ";
00612                }
00613                else
00614                {
00615                    std::cout « f.code;
00616                }
00617            }
00618            std::cout « std::endl;
00619        }
00620    }
00621 }
```

4.11.1.38 showPath() [2/2]

```
void showPath (
            vector< vector< Flight > > paths )
```

Function to show a set of paths Complexity: O(n), where n is the total number of flights in all paths.

Parameters

paths	A vector of vectors of flights representing the paths.

Returns

void

Definition at line 632 of file dbairport.cpp.

```
00633 {
00634     if (paths.empty())
00635     {
00636         std::cout « "-------------------------------------------------"
00637                   « std::endl;
00638         std::cout « "Sorry, but there is no result with those inputs."
00639                   « std::endl;
00640         return;
00641     }
00642
00643     for (auto path : paths)
00644     {
00645         std::cout « "-------------------------------------------------"
00646                   « std::endl;
00647         for (auto p : path)
00648         {
00649             if (!p.airline.empty())
00650             {
00651                 std::cout « p.code « " -(" « p.airline « ")"
00652                           « "-> ";
00653             }
00654             else
00655             {
00656                 std::cout « p.code;
00657             }
00658         }
00659         std::cout « std::endl;
00660     }
00661 }
```

4.11.1.39 toRadians()

```
double toRadians (
            const double degree )
```

Function to convert Degrees in radians Complexity: O(1)

Parameters

degree	The degree to be converted.

Returns

double The converted degree.

Definition at line 718 of file dbairport.cpp.

```
00718 { return (degree * M_PI / 180); }
```

4.12 dbairport.cpp

Go to the documentation of this file.

```
00001 #include "dbairport.h"
00002
00003 //------------------------------------------------------------
00004
00011 int quantityAirports(Graph<Airport> airports)
00012 {
00013     return airports.getNumVertex();
00014 }
00015
00023 int quantityAirportsCountry(Graph<Airport> airports, std::string country)
00024 {
00025     int count = 0;
00026     for (auto v : airports.getVertexSet())
00027     {
00028         Airport a = v->getInfo();
00029         if (a.getCountry() == country)
00030         {
00031             count++;
00032         }
00033     }
00034     return count;
00035 }
00036
00044 int quantityAirportsCity(Graph<Airport> airports, std::string city)
00045 {
00046     int count = 0;
00047     for (auto v : airports.getVertexSet())
00048     {
00049         Airport a = v->getInfo();
00050         if (a.getCity() == city)
00051         {
00052             count++;
00053         }
00054     }
00055     return count;
00056 }
00057
00065 int quantityCitiesCountry(Graph<Airport> airports, std::string country)
00066 {
00067     std::unordered_set<std::string> uniqueCities;
00068     for (auto v : airports.getVertexSet())
00069     {
00070         Airport a = v->getInfo();
00071         if (a.getCountry() == country)
00072         {
00073             std::string city = a.getCity();
00074             uniqueCities.insert(city);
00075         }
00076     }
00077     return uniqueCities.size();
00078 }
00079
00080 //------------------------------------------------------------
00081
00090 int quantityAirlinesCountry(unordered_map<string, Airline> airlines, std::string country)
00091 {
00092     int count = 0;
00093     for (auto v : airlines)
00094     {
00095         if (v.second.getCountry() == country)
```

```
00096         {
00097             count++;
00098         }
00099     }
00100     return count;
00101 }
00102
00103 //-------------------------------------------------------------
00104
00111 int quantityFlights(Graph<Airport> airports)
00112 {
00113     int count = 0;
00114     for (auto v : airports.getVertexSet())
00115     {
00116         count += v->getAdj().size();
00117     }
00118     return count;
00119 }
00120
00128 int quantityFlightsAirport(Graph<Airport> airports, std::string airport)
00129 {
00130     int count = 0;
00131     for (auto v : airports.getVertexSet())
00132     {
00133         if (v->getInfo().getCode() == airport)
00134         {
00135             count += v->getAdj().size();
00136         }
00137     }
00138     return count;
00139 }
00140
00148 int quantityFlightsCountry(Graph<Airport> airports, std::string country)
00149 {
00150     int count = 0;
00151     for (auto v : airports.getVertexSet())
00152     {
00153         if (v->getInfo().getCountry() == country)
00154         {
00155             count += v->getAdj().size();
00156         }
00157     }
00158     return count;
00159 }
00160
00161 // Update later need country too cause have cities with same name
00169 int quantityFlightsCity(Graph<Airport> airports, std::string city)
00170 {
00171     int count = 0;
00172     for (auto v : airports.getVertexSet())
00173     {
00174         if (v->getInfo().getCity() == city)
00175         {
00176             count += v->getAdj().size();
00177         }
00178     }
00179     return count;
00180 }
00181
00190 int quantityFlightsAirline(Graph<Airport> airports, std::string airline)
00191 {
00192     int count = 0;
00193     for (auto v : airports.getVertexSet())
00194     {
00195         for (auto e : v->getAdj())
00196         {
00197             if (e.getRoute() == airline)
00198             {
00199                 count++;
00200             }
00201         }
00202     }
00203
00204     return count;
00205 }
00206
00207 //-------------------------------------------------------------
00208
00216 int quantityDestinationsAirport(Graph<Airport> airports, std::string airport)
00217 {
00218     std::set<std::string> countries;
00219     std::vector<std::string> res;
00220     resetVisited(airports);
00221     auto s = airports.findVertex(Airport(airport));
00222     if (s == nullptr)
00223         return 0;
00224     Graph<Airport> g;
```

```
00225     res = dfsVisit(s, res);
00226     for (auto c : res)
00227     {
00228         countries.insert(c);
00229     }
00230     return countries.size();
00231 }
00232
00241 int quantityDestinationLimitedStop(Graph<Airport> airports, std::string airport,
00242                                    int stop)
00243 {
00244     std::set<std::string> countries;
00245     std::vector<std::string> res;
00246     resetVisited(airports);
00247     auto s = airports.findVertex(Airport(airport));
00248     if (s == nullptr)
00249         return 0;
00250     res = dfsVisit(s, res, (stop + 1));
00251     for (auto c : res)
00252     {
00253         countries.insert(c);
00254     }
00255     return countries.size();
00256 }
00257
00265 int quantityDestinationMax(Graph<Airport> airports)
00266 {
00267     vector<string> path, maxPath;
00268
00269     for (auto v : airports.getVertexSet())
00270     {
00271         // std::cout << "Consultando... " << v->getInfo().getCode() << std::endl;
00272         resetVisited(airports);
00273         path.clear();
00274         dfsMax(v, path, maxPath);
00275     }
00276
00277     std::cout << "\nStarting in: " << maxPath[0] << std::endl;
00278     // for (int i = 1; i < maxPath.size() - 1; i++)
00279     // {
00280     //     std::cout << maxPath[i] << " -> ";
00281     // }
00282     std::cout << "\nEnding in: " << maxPath[maxPath.size() - 1] << std::endl;
00283     return maxPath.size() - 1;
00284 }
00285
00293 void dfsMax(Vertex<Airport> *v, std::vector<std::string> &path, std::vector<std::string> &maxPath)
00294 {
00295     v->setVisited(true);
00296     path.push_back(v->getInfo().getCode());
00297     auto adjs = v->getAdj();
00298
00299     if (path.size() > maxPath.size())
00300     {
00301         maxPath = path;
00302     }
00303
00304     for (auto &e : adjs)
00305     {
00306         auto w = e.getDest();
00307
00308         if (!w->isVisited())
00309         {
00310             dfsMax(w, path, maxPath);
00311         }
00312     }
00313     path.pop_back();
00314 }
00315
00316 //------------------------------------------------------------
00317 //------------------------------------------------------------
00318 //------------------------------------------------------------
00319
00326 bool comparatorPath(const vector<Flight> a, const vector<Flight> b)
00327 {
00328     return a.size() < b.size();
00329 }
00330
00338 std::pair<int, int> quantityFlights(Graph<Airport> airports, std::string code)
00339 {
00340     int count = 0;
00341     std::set<std::string> airlines;
00342
00343     auto s = airports.findVertex(Airport(code));
00344     if (s != nullptr)
00345     {
00346         for (auto v : s->getAdj())
```

```
00347         {
00348             count++;
00349             airlines.insert(v.getRoute());
00350         }
00351     }
00352     return std::pair<int, int>(count, airlines.size());
00353 }
00354
00361 std::vector<std::string> dfsVisit(Vertex<Airport> *v,
00362                                   std::vector<std::string> &res)
00363 {
00364     v->setVisited(true);
00365     res.push_back(v->getInfo().getCountry());
00366     auto adjs = v->getAdj();
00367
00368     for (auto &e : adjs)
00369     {
00370         auto w = e.getDest();
00371
00372         if (!w->isVisited())
00373         {
00374             dfsVisit(w, res);
00375         }
00376     }
00377
00378     return res;
00379 }
00380
00389 std::vector<std::string> dfsVisit(Vertex<Airport> *v,
00390                                   std::vector<std::string> &res, int stop)
00391 {
00392     if (stop == 0)
00393         return res;
00394
00395     // std::cout << "Saindo de " << v->getInfo().getCode() << std::endl;
00396
00397     v->setVisited(true);
00398     res.push_back(v->getInfo().getCountry());
00399     auto adjs = v->getAdj();
00400
00401     for (auto &e : adjs)
00402     {
00403         auto w = e.getDest();
00404
00405         if (!w->isVisited())
00406         {
00407             // std::cout << "\t Eu vou para: " << w->getInfo().getCode() <<
00408             // std::endl;
00409             dfsVisit(w, res, (stop - 1));
00410         }
00411     }
00412     return res;
00413 }
00414
00421 void resetVisited(Graph<Airport> &airports)
00422 {
00423     for (auto v : airports.getVertexSet())
00424     {
00425         v->setVisited(false);
00426     }
00427 }
00428
00436 void calculateIndegree(Graph<Airport> &airports)
00437 {
00438     for (auto v : airports.getVertexSet())
00439     {
00440         v->setIndegree(0);
00441     }
00442     for (auto v : airports.getVertexSet())
00443     {
00444         for (auto e : v->getAdj())
00445         {
00446             e.getDest()->setIndegree(e.getDest()->getIndegree() + 1);
00447         }
00448     }
00449 }
00450
00460 void rankingAirports(Graph<Airport> airports, int arg)
00461 {
00462     std::vector<Ranking> vec;
00463     calculateIndegree(airports);
00464
00465     for (auto v : airports.getVertexSet())
00466     {
00467         int total = v->getIndegree() + v->getAdj().size();
00468         Ranking rank = {v->getInfo().getCode(), total};
00469         vec.push_back(rank);
```

```
00470     }
00471     std::sort(vec.begin(), vec.end(), comparator);
00472     int i = 0;
00473     for (auto v : vec)
00474     {
00475         if (i < arg)
00476         {
00477             std::cout « "Code: " « v.code « " / ";
00478             std::cout « "Name: " « airportsHash.find(v.code)->second.getName() « " / ";
00479             std::cout « "Total: " « v.count « std::endl;
00480             i++;
00481         }
00482     }
00483 }
00484
00494 void findBestFlights(Graph<Airport> &airports, string src, string dest,
00495                      vector<string> &airplanes)
00496 {
00497     resetVisited(airports);
00498
00499     auto s = airports.findVertex(Airport(src));
00500     auto d = airports.findVertex(Airport(dest));
00501
00502     if (s == nullptr || d == nullptr)
00503         return;
00504
00505     vector<vector<Flight» paths;
00506     paths = bfsPath(s, dest, airplanes);
00507     showPath(paths);
00508 }
00509
00522 void findBestFlights(Graph<Airport> &airports, string country, string city,
00523                      string airport, int type, vector<string> &airplanes)
00524 {
00525     vector<Vertex<Airport> *> vec;
00526     vector<vector<Flight» paths;
00527     vec = findAirports(airports, country, city);
00528
00529     if (type == 0)
00530     {
00531         for (auto v : vec)
00532         {
00533             // std::cout « "Source: " « v->getInfo().getCode() « std::endl;
00534             resetVisited(airports);
00535             paths = bfsPath(v, airport, airplanes);
00536             showPath(paths);
00537         }
00538     }
00539     else if (type == 1)
00540     {
00541         for (auto v : vec)
00542         {
00543             // std::cout « "Source: " « airport « std::endl;
00544             std::string tgt = v->getInfo().getCode();
00545             resetVisited(airports);
00546             paths = bfsPath(airports.findVertex(Airport(airport)), tgt, airplanes);
00547             showPath(paths);
00548         }
00549     }
00550 }
00551
00563 void showPath(Graph<Airport> &airports, vector<Vertex<Airport> *> source,
00564               vector<Vertex<Airport> *> dest, vector<vector<Flight» paths,
00565               vector<string> &airplanes)
00566 {
00567
00568     vector<vector<Flight» flights;
00569
00570     for (auto s : source)
00571     {
00572         // std::cout « "Source: " « s->getInfo().getCode() « std::endl;
00573         for (auto d : dest)
00574         {
00575             // std::cout « "Destination: " « d->getInfo().getCode() «
00576             // std::endl;
00577             std::string tgt = d->getInfo().getCode();
00578             resetVisited(airports);
00579             flights = bfsPath(s, tgt, airplanes);
00580             paths.insert(paths.end(), flights.begin(), flights.end());
00581         }
00582     }
00583
00584     if (paths.empty())
00585     {
00586         std::cout « "-------------------------------------------------"
00587                   « std::endl;
00588         std::cout « "Sorry, but there is no result with those inputs."
```

```
00589                   « std::endl;
00590         return;
00591     }
00592     else
00593     {
00594
00595         std::sort(paths.begin(), paths.end(), comparatorPath);
00596         const int min = paths[0].size();
00597
00598         for (auto &p : paths)
00599         {
00600             if (p.size() > min)
00601             {
00602                 continue;
00603             }
00604             std::cout « "-------------------------------------------------"
00605                       « std::endl;
00606             for (auto &f : p)
00607             {
00608                 if (!f.airline.empty())
00609                 {
00610                     std::cout « f.code « " -(" « f.airline « ")"
00611                               « "-> ";
00612                 }
00613                 else
00614                 {
00615                     std::cout « f.code;
00616                 }
00617             }
00618             std::cout « std::endl;
00619         }
00620     }
00621 }
00622
00623 // Function to show a set of paths
00624 // Complexity: O(n), where n is the total number of flights in all paths
00625
00632 void showPath(vector<vector<Flight» paths)
00633 {
00634     if (paths.empty())
00635     {
00636         std::cout « "-------------------------------------------------"
00637                   « std::endl;
00638         std::cout « "Sorry, but there is no result with those inputs."
00639                   « std::endl;
00640         return;
00641     }
00642
00643     for (auto path : paths)
00644     {
00645         std::cout « "-------------------------------------------------"
00646                   « std::endl;
00647         for (auto p : path)
00648         {
00649             if (!p.airline.empty())
00650             {
00651                 std::cout « p.code « " -(" « p.airline « ")"
00652                           « "-> ";
00653             }
00654             else
00655             {
00656                 std::cout « p.code;
00657             }
00658         }
00659         std::cout « std::endl;
00660     }
00661 }
00662
00671 vector<Vertex<Airport> *> findAirports(Graph<Airport> &airports, string country,
00672                                        string city)
00673 {
00674     vector<Vertex<Airport> *> vec;
00675     for (auto v : airports.getVertexSet())
00676     {
00677         if (v->getInfo().getCountry() == country &&
00678             v->getInfo().getCity() == city)
00679         {
00680             vec.push_back(v);
00681         }
00682     }
00683     return vec;
00684 }
00685
00698 void findBestFlights(Graph<Airport> &airports, string countrySrc,
00699                      string citySrc, string countryDest, string cityDest,
00700                      vector<string> &airplanes)
00701 {
```

```
00702     vector<Vertex<Airport> *> src;
00703     vector<Vertex<Airport> *> dest;
00704     vector<vector<Flight» paths;
00705
00706     src = findAirports(airports, countrySrc, citySrc);
00707     dest = findAirports(airports, countryDest, cityDest);
00708
00709     showPath(airports, src, dest, paths, airplanes);
00710 }
00711
00718 double toRadians(const double degree) { return (degree * M_PI / 180); }
00719
00720 //-23.477461, -46.548338 Anywhere in Sao Paulo
00721 //-23.432075, -46.469511 Garulhos GRU 10km
00722
00723 // 28.160090, -17.240129 Anywhere in La Gomera
00724 // 28.029600, -17.214600 GMZ 30km
00725
00735 double distanceEarth(double latOrigin, double longOrigin, double latDest,
00736                      double longDest)
00737 {
00738     latOrigin = toRadians(latOrigin);
00739     longOrigin = toRadians(longOrigin);
00740     latDest = toRadians(latDest);
00741     longDest = toRadians(longDest);
00742
00743     double dlong = longDest - longOrigin;
00744     double dlat = latDest - latOrigin;
00745
00746     double ans = pow(sin(dlat / 2), 2) +
00747                  cos(latOrigin) * cos(latDest) * pow(sin(dlong / 2), 2);
00748
00749     ans = 2 * asin(sqrt(ans));
00750
00751     double R = 6371;
00752
00753     ans = ans * R;
00754
00755     return ans;
00756 }
00757
00758 // Function to find all airports around a point withing a distance
00759 // Complexity: O(n), where n is the number of airports in the graph
00760
00770 vector<Vertex<Airport> *> findAirports(Graph<Airport> &airports, double lat,
00771                                        double lon, int distMax)
00772 {
00773     vector<Vertex<Airport> *> vec;
00774     for (auto v : airports.getVertexSet())
00775     {
00776         double lat2 = v->getInfo().getLatitude();
00777         double lon2 = v->getInfo().getLongitude();
00778
00779         double dist = distanceEarth(lat, lon, lat2, lon2);
00780
00781         if (dist <= distMax)
00782         {
00783             vec.push_back(v);
00784         }
00785     }
00786     return vec;
00787 }
00788
00802 void findBestFlights(Graph<Airport> &airports, double latOrigin,
00803                      double longOrigin, double latDest, double longDest,
00804                      int distMax, vector<string> &airplanes)
00805 {
00806     vector<Vertex<Airport> *> src =
00807         findAirports(airports, latOrigin, longOrigin, distMax);
00808     vector<Vertex<Airport> *> dest =
00809         findAirports(airports, latDest, longDest, distMax);
00810     vector<vector<Flight» paths;
00811
00812     showPath(airports, src, dest, paths, airplanes);
00813 }
00814
00828 void findBestFlights(Graph<Airport> &airports, string airport, double lat,
00829                      double lon, int distMax, int type,
00830                      vector<string> &airplanes)
00831 {
00832     vector<Vertex<Airport> *> vec;
00833     vector<vector<Flight» paths;
00834     vec = findAirports(airports, lat, lon, distMax);
00835
00836     if (type == 0)
00837     {
00838         for (auto v : vec)
```

```
00839         {
00840             std::cout « "Source: " « v->getInfo().getCode() « std::endl;
00841             resetVisited(airports);
00842             paths = bfsPath(v, airport, airplanes);
00843             showPath(paths);
00844         }
00845     }
00846     else if (type == 1)
00847     {
00848         for (auto v : vec)
00849         {
00850             std::cout « "Source: " « airport « std::endl;
00851             std::string tgt = v->getInfo().getCode();
00852             resetVisited(airports);
00853             paths = bfsPath(airports.findVertex(Airport(airport)), tgt, airplanes);
00854             showPath(paths);
00855         }
00856     }
00857 }
00858
00859 // Function to find best flights between a city and a point on the Earth
00860 // type = 0: City to Point, type = 1: Point to City
00861 // Complexity: O(n^2 + n * e), where n is the number of airports and e is
00862 // the number of flights in the graph
00863
00878 void findBestFlights(Graph<Airport> &airports, string country, string city,
00879                      double lat, double lon, int distMax, int type,
00880                      vector<string> &airplanes)
00881 {
00882     vector<Vertex<Airport> *> vec_point;
00883     vector<Vertex<Airport> *> vec_city;
00884     vector<vector<Flight» paths;
00885
00886     vec_point = findAirports(airports, lat, lon, distMax);
00887     vec_city = findAirports(airports, country, city);
00888
00889     if (type == 0)
00890     {
00891         for (auto c : vec_city)
00892         {
00893             std::cout « "Source: " « c->getInfo().getCode() « std::endl;
00894             for (auto p : vec_point)
00895             {
00896                 std::cout « "Destination: " « p->getInfo().getCode() « std::endl;
00897                 resetVisited(airports);
00898                 std::string tgt = p->getInfo().getCode();
00899                 paths = bfsPath(c, tgt, airplanes);
00900                 showPath(paths);
00901             }
00902         }
00903     }
00904     else if (type == 1)
00905     {
00906         for (auto p : vec_point)
00907         {
00908             std::cout « "Source: " « p->getInfo().getCode() « std::endl;
00909             for (auto c : vec_city)
00910             {
00911                 std::cout « "Destination: " « c->getInfo().getCode() « std::endl;
00912                 resetVisited(airports);
00913                 std::string tgt = c->getInfo().getCode();
00914                 paths = bfsPath(p, tgt, airplanes);
00915                 showPath(paths);
00916             }
00917         }
00918     }
00919 }
00920
00932 void getPath(string current, vector<Flight> &path,
00933              unordered_map<string, vector<Flight» &prev,
00934              vector<vector<Flight» &paths, string startCode, string airline)
00935 {
00936     path.push_back({current, airline});
00937
00938     if (current == startCode)
00939     {
00940         vector<Flight> validPath = path;
00941         reverse(validPath.begin(), validPath.end());
00942         paths.push_back(validPath);
00943     }
00944     else
00945     {
00946         for (auto &prevVertex : prev[current])
00947         {
00948             getPath(prevVertex.code, path, prev, paths, startCode,
00949                     prevVertex.airline);
00950         }
```

```
00951     }
00952
00953     path.pop_back();
00954 }
00955
00964 vector<vector<Flight» bfsPath(Vertex<Airport> *v, string &tgt,
00965                               vector<string> &airplanes)
00966 {
00967     vector<vector<Flight» paths;
00968     unordered_map<string, vector<Flight» prev;
00969     unordered_map<string, int> dist;
00970     queue<Vertex<Airport> *> q;
00971
00972     q.push(v);
00973     v->setVisited(true);
00974     dist[v->getInfo().getCode()] = 0;
00975
00976     while (!q.empty())
00977     {
00978         auto vertex = q.front();
00979         q.pop();
00980         auto adjs = vertex->getAdj();
00981
00982         for (auto &e : adjs)
00983         {
00984             auto w = e.getDest();
00985             if (!w->isVisited() && (find(airplanes.begin(), airplanes.end(),
00986                                          e.getRoute()) != airplanes.end() ||
00987                                     airplanes.size() == 0))
00988             {
00989                 q.push(w);
00990                 w->setVisited(true);
00991                 prev[w->getInfo().getCode()].push_back(
00992                     {vertex->getInfo().getCode(), e.getRoute()});
00993                 dist[w->getInfo().getCode()] = dist[vertex->getInfo().getCode()] + 1;
00994             }
00995             else if (dist[w->getInfo().getCode()] ==
00996                          dist[vertex->getInfo().getCode()] + 1 &&
00997                      (find(airplanes.begin(), airplanes.end(), e.getRoute()) !=
00998                           airplanes.end() ||
00999                       airplanes.size() == 0))
01000             {
01001                 prev[w->getInfo().getCode()].push_back(
01002                     {vertex->getInfo().getCode(), e.getRoute()});
01003             }
01004         }
01005     }
01006
01007     vector<Flight> path;
01008     getPath(tgt, path, prev, paths, v->getInfo().getCode(), "");
01009
01010     return paths;
01011 }
01012
01019 int connectedComponents(Graph<Airport> &airports)
01020 {
01021     int component = 0;
01022     resetVisited(airports);
01023     Graph<Airport> g = airports;
01024
01025     for (auto v : g.getVertexSet())
01026     {
01027         if (!v->isVisited())
01028         {
01029             dfsConnectedComponents(g, v);
01030             component++;
01031         }
01032     }
01033     // std::cout « "Number of connected components: " « component «
01034     // std::endl;
01035     return component;
01036 }
01037
01045 void dfsConnectedComponents(Graph<Airport> &airports, Vertex<Airport> *v)
01046 {
01047     v->setVisited(true);
01048     auto adjs = v->getAdj();
01049
01050     for (auto &e : adjs)
01051     {
01052         auto w = e.getDest();
01053         if (!w->isVisited())
01054         {
01055             dfsConnectedComponents(airports, w);
01056         }
01057     }
01058 }
```

```
01059
01066 void findArticulationPoints(Graph<Airport> &airports)
01067 {
01068     int i = 0;
01069     int total = 0;
01070     int connected = connectedComponents(airports);
01071     for (auto v : airports.getVertexSet())
01072     {
01073         // std::cout « " Analisando o vertice: " « v->getInfo().getCode() «
01074         // std::endl;
01075         resetVisited(airports);
01076         int component = 0;
01077         for (auto w : airports.getVertexSet())
01078         {
01079             // std::cout « "\tAnalisando o vertice: " « w->getInfo().getCode()«
01080             // std::endl;
01081             if (!w->isVisited() && w->getInfo().getCode() != v->getInfo().getCode())
01082             {
01083                 // std::cout « "\t\tChamando o DFS component++" « std::endl;
01084                 dfsArtc(v, w);
01085                 component++;
01086             }
01087         }
01088         if (component > connected)
01089         {
01090             std::cout « v->getInfo().getCode() « " ";
01091             i++;
01092             total++;
01093             if (i == 10)
01094             {
01095                 std::cout « std::endl;
01096                 i = 0;
01097             }
01098         }
01099     }
01100     std::cout « "\nTotal: " « total « std::endl;
01101 }
01102
01109 void dfsArtc(Vertex<Airport> *v, Vertex<Airport> *w)
01110 {
01111     w->setVisited(true);
01112     auto adjs = w->getAdj();
01113
01114     for (auto &e : adjs)
01115     {
01116         auto t = e.getDest();
01117         if (!t->isVisited() && t->getInfo().getCode() != v->getInfo().getCode())
01118         {
01119             dfsArtc(v, t);
01120         }
01121     }
01122 }
```

4.13 src/database/dbairport.h File Reference

```
#include "../classes/Airline.h"
#include "../classes/Airport.h"
#include "../classes/Graph.h"
#include "read.h"
#include <algorithm>
#include <cmath>
#include <fstream>
#include <limits.h>
#include <map>
#include <set>
#include <unordered_map>
#include <unordered_set>
```

Classes

- struct Flight

Functions

- int quantityAirports (Graph< Airport > airports)

  Function to get quantity of airports in the graph Complexity: O(1)
- int quantityAirportsCountry (Graph< Airport > airports, std::string country)

  Function to get quantity of airports in the graph by country Complexity: O(n), where n is the number of airports in the graph.
- int quantityAirportsCity (Graph< Airport > airports, std::string city)

  Function to get quantity of airports in the graph by city Complexity: O(n), where n is the number of airports in the graph.
- int quantityCitiesCountry (Graph< Airport > airports, std::string country)

  Function to count the number of unique cities in the graph for a given country Complexity: O(n), where n is the number of airports in the graph.
- int quantityAirlinesCountry (unordered_map< string, Airline > airlines, std::string country)

  Function to calculate the quantity of airlines for a given country Complexity: O(n), where n is the number of airlines in the graph.
- int quantityFlights (Graph< Airport > airports)

  Function to calculate the quatity of flights in the graph Complexity: O(n), where n is the number of airports in the graph.
- int quantityFlightsAirport (Graph< Airport > airports, std::string airport)

  Function to calculate the quatity of flights in a given airport Complexity: O(n), where n is the number of airports in the graph.
- int quantityFlightsCountry (Graph< Airport > airports, std::string country)

  Function to calculate the quatity of flights in a given country Complexity: O(n), where n is the number of airports in the graph.
- int quantityFlightsCity (Graph< Airport > airports, std::string city)

  Function to calculate the quatity of flights in a given city Complexity: O(n), where n is the number of airports in the graph.
- int quantityFlightsAirline (Graph< Airport > airports, std::string airline)

  Function to calculate the quatity of flights in a given airline Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.
- int quantityDestinationsAirport (Graph< Airport > airports, std::string airport)

  Function to calculate the number of different countries it is possible to go directly to from an airport Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.
- int quantityDestinationLimitedStop (Graph< Airport > airports, std::string airport, int stop)

  Function to calculate the number of different countries it is possible to go to from an airport with a given number of stops Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.
- int quantityDestinationMax (Graph< Airport > airports)

  Function that calculates the maximum possible trip, with the greatest number of stops and print the starting airport and the destination airport Complexity: O(n ∗ (n + e)), where n is the number of airports in the graph and e is the number of flights.
- std::pair< int, int > quantityFlights (Graph< Airport > airports, std::string code)

  Function to get quantity of flights in the graph and unique airlines for an Airport Complexity: O(n), where n is the number of flights in the graph.
- std::vector< std::string > dfsVisit (Vertex< Airport > ∗v, std::vector< std::string > &res)

  Search (depth-first) to visit vertices and collect connected countries.
- std::vector< std::string > dfsVisit (Vertex< Airport > ∗v, std::vector< std::string > &res, int stop)

  Search (depth-first) to visit vertices and collect connected countries withing a number of stops.
- void resetVisited (Graph< Airport > &airports)

  Function to reset the visited status of all vertices in the graph Complexity: O(n), where n is the number of airports in the graph.
- void rankingAirports (Graph< Airport > airports, int arg)

  Function to rank airport based of the sum of in-degrees and out-degrees Complexity: O(n ∗ log(n) + n ∗ e), where n is the number of airports and e is the number of flights in the graph (assuming sorting has a time complexity of O(n ∗ log(n)))

- void calculateIndegree (Graph< Airport > &airports)

  Function to calculate the in-degreee of each vertex in the graph Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findBestFlights (Graph< Airport > &airports, string src, string dest, vector< string > &airplanes)

  Function to find the best flights from a source airport to a destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findBestFlights (Graph< Airport > &airports, string countrySrc, string citySrc, string countryDest, string cityDest, vector< string > &airplanes)

  Function to find best flights between two cities This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findBestFlights (Graph< Airport > &airports, double latOrigin, double longOrigin, double latDest, double longDest, int distMax, vector< string > &airplanes)

  Function to find the best flights between two points on Earth Point to Point This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findBestFlights (Graph< Airport > &airports, string country, string city, string airport, int type, vector< string > &airplanes)

  Finds the best flights between a city and an aiport (vice-verse) This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void findBestFlights (Graph< Airport > &airports, string country, string city, double lat, double lon, int distMax, int type, vector< string > &airplanes)

  Function to find best flights between a city and a point on the Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p∗(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.
- void findBestFlights (Graph< Airport > &airports, string airport, double lat, double lon, int distMax, int type, vector< string > &airplanes)

  Function to find best flights between an airport and a point on Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p∗(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.
- void showPath (vector< vector< Flight > > paths)

  Function to show a set of paths Complexity: O(n), where n is the total number of flights in all paths.
- void showPath (Graph< Airport > &airports, vector< Vertex< Airport > ∗ > source, vector< Vertex< Airport > ∗ > dest, vector< vector< Flight > > paths, vector< string > &airplanes)

  Function to show a multi-path between multiple source and destination airports Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- vector< Vertex< Airport > ∗ > findAirports (Graph< Airport > &airports, string country, string city)

  Function to find all airports in a city Complexity: O(n), where n is the number of airports in the graph.
- vector< Vertex< Airport > ∗ > findAirports (Graph< Airport > &airports, double lat, double lon, int distMax)

  Function to find all airports around a point withing a distance Complexity: O(n), where n is the number of airports in the graph.
- vector< vector< Flight > > bfsPath (Vertex< Airport > ∗v, string &tgt, vector< string > &airplanes)

  Function to perfom breadth-first search to find paths between two airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.
- void getPath (string current, vector< Flight > &path, unordered_map< string, vector< Flight > > &prev, vector< vector< Flight > > &paths, string startCode, string airline)

  Function to find a path from the current airport to start airport Complexity: O(n), where n is the number of airports in the graph.
- void dfsArtc (Vertex< Airport > ∗v, Vertex< Airport > ∗w)

  Function DFS modified to find articulation points.
- void findArticulationPoints (Graph< Airport > &airports)

  Function to find and show articulation points in the graph. Points (airports) that, if removed, make certain regions of the map inaccessible Complexity: O(n^2), where n is the number of airports in the graph.
- bool comparatorPath (const vector< Flight > a, const vector< Flight > b)

Function to sort the vector with the best flights in increasing order by the number of stops.

- void dfsConnectedComponents (Graph< Airport > &airports, Vertex< Airport > *v)

  Function DFS starting from a vertex to find connected components Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

- int connectedComponents (Graph< Airport > &airports)

  Function that uses a depth-first search to find the number of connected components in a graph. Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

- void dfsMax (Vertex< Airport > *v, std::vector< std::string > &path, std::vector< std::string > &maxPath)

  Function that performs a depth-first search and always updates the longest path.

- double toRadians (const double degree)

  Function to convert Degrees in radians Complexity: O(1)

- double distanceEarth (double latOrigin, double longOrigin, double latDest, double longDest)

  Function to calculate the distance between two points on the Earth Complexity: O(1)

4.13.1 Function Documentation

4.13.1.1 bfsPath()

```
vector< vector< Flight > > bfsPath (
            Vertex< Airport > * v,
            string & tgt,
            vector< string > & airplanes )
```

Function to perfom breadth-first search to find paths between two airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

v	The vertex to start the search
tgt	The target airport.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

vector<vector<Flight>> A vector of vectors of flights representing the paths.

Definition at line 964 of file dbairport.cpp.

```
00966 {
00967     vector<vector<Flight>> paths;
00968     unordered_map<string, vector<Flight>> prev;
00969     unordered_map<string, int> dist;
00970     queue<Vertex<Airport> *> q;
00971
00972     q.push(v);
00973     v->setVisited(true);
00974     dist[v->getInfo().getCode()] = 0;
00975
00976     while (!q.empty())
00977     {
00978         auto vertex = q.front();
00979         q.pop();
00980         auto adjs = vertex->getAdj();
00981
00982         for (auto &e : adjs)
00983         {
00984             auto w = e.getDest();
00985             if (!w->isVisited() && (find(airplanes.begin(), airplanes.end(),
00986                                          e.getRoute()) != airplanes.end() ||
00987                                     airplanes.size() == 0))
```

```
00988             {
00989                 q.push(w);
00990                 w->setVisited(true);
00991                 prev[w->getInfo().getCode()].push_back(
00992                     {vertex->getInfo().getCode(), e.getRoute()});
00993                 dist[w->getInfo().getCode()] = dist[vertex->getInfo().getCode()] + 1;
00994             }
00995             else if (dist[w->getInfo().getCode()] ==
00996                         dist[vertex->getInfo().getCode()] + 1 &&
00997                      (find(airplanes.begin(), airplanes.end(), e.getRoute()) !=
00998                           airplanes.end() ||
00999                       airplanes.size() == 0))
01000             {
01001                 prev[w->getInfo().getCode()].push_back(
01002                     {vertex->getInfo().getCode(), e.getRoute()});
01003             }
01004         }
01005     }
01006
01007     vector<Flight> path;
01008     getPath(tgt, path, prev, paths, v->getInfo().getCode(), "");
01009
01010     return paths;
01011 }
```

4.13.1.2 calculateIndegree()

```
void calculateIndegree (
            Graph< Airport > & airports )
```

Function to calculate the in-degreee of each vertex in the graph Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph of airports.

Returns

void

Definition at line 436 of file dbairport.cpp.

```
00437 {
00438     for (auto v : airports.getVertexSet())
00439     {
00440         v->setIndegree(0);
00441     }
00442     for (auto v : airports.getVertexSet())
00443     {
00444         for (auto e : v->getAdj())
00445         {
00446             e.getDest()->setIndegree(e.getDest()->getIndegree() + 1);
00447         }
00448     }
00449 }
```

4.13.1.3 comparatorPath()

```
bool comparatorPath (
            const vector< Flight > a,
            const vector< Flight > b )
```

Function to sort the vector with the best flights in increasing order by the number of stops.

Parameters

a	The first vector.
b	The second vector.

Returns

bool True if the first vector is smaller than the second vector.

Definition at line 326 of file dbairport.cpp.

```
00327 {
00328     return a.size() < b.size();
00329 }
```

4.13.1.4 connectedComponents()

```
int connectedComponents (
            Graph< Airport > & airports )
```

Function that uses a depth-first search to find the number of connected components in a graph. Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.

Returns

int The number of connected components.

Definition at line 1019 of file dbairport.cpp.

```
01020 {
01021     int component = 0;
01022     resetVisited(airports);
01023     Graph<Airport> g = airports;
01024
01025     for (auto v : g.getVertexSet())
01026     {
01027         if (!v->isVisited())
01028         {
01029             dfsConnectedComponents(g, v);
01030             component++;
01031         }
01032     }
01033     // std::cout « "Number of connected components: " « component «
01034     // std::endl;
01035     return component;
01036 }
```

4.13.1.5 dfsArtc()

```
void dfsArtc (
            Vertex< Airport > * v,
            Vertex< Airport > * w )
```

Function DFS modified to find articulation points.

Parameters

v	The vertex to start the search
w	The vertex to be visited

Returns

void

Definition at line 1109 of file dbairport.cpp.

```
01110 {
01111     w->setVisited(true);
01112     auto adjs = w->getAdj();
01113
01114     for (auto &e : adjs)
01115     {
01116         auto t = e.getDest();
01117         if (!t->isVisited() && t->getInfo().getCode() != v->getInfo().getCode())
01118         {
01119             dfsArtc(v, t);
01120         }
01121     }
01122 }
```

4.13.1.6 dfsConnectedComponents()

```
void dfsConnectedComponents (
            Graph< Airport > & airports,
            Vertex< Airport > * v )
```

Function DFS starting from a vertex to find connected components Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
v	The vertex to start the search

Returns

void

Definition at line 1045 of file dbairport.cpp.

```
01046 {
01047     v->setVisited(true);
01048     auto adjs = v->getAdj();
01049
01050     for (auto &e : adjs)
01051     {
01052         auto w = e.getDest();
01053         if (!w->isVisited())
01054         {
01055             dfsConnectedComponents(airports, w);
01056         }
01057     }
01058 }
```

4.13.1.7 dfsMax()

```
void dfsMax (
            Vertex< Airport > * v,
```

```
            std::vector< std::string > & path,
            std::vector< std::string > & maxPath )
```

Function that performs a depth-first search and always updates the longest path.

Parameters

v	The vertex to start the search
path	The current path
maxPath	The longest path

Returns

void

Definition at line 293 of file dbairport.cpp.

```
00294 {
00295     v->setVisited(true);
00296     path.push_back(v->getInfo().getCode());
00297     auto adjs = v->getAdj();
00298 
00299     if (path.size() > maxPath.size())
00300     {
00301         maxPath = path;
00302     }
00303 
00304     for (auto &e : adjs)
00305     {
00306         auto w = e.getDest();
00307 
00308         if (!w->isVisited())
00309         {
00310             dfsMax(w, path, maxPath);
00311         }
00312     }
00313     path.pop_back();
00314 }
```

4.13.1.8 dfsVisit() [1/2]

```
std::vector< std::string > dfsVisit (
            Vertex< Airport > * v,
            std::vector< std::string > & res )
```

Search (depth-first) to visit vertices and collect connected countries.

Parameters

airports	The graph of airports.
v	The vertex to start the search

Returns

vector<string> The vector with the countries.

Definition at line 361 of file dbairport.cpp.

```
00363 {
00364     v->setVisited(true);
00365     res.push_back(v->getInfo().getCountry());
00366     auto adjs = v->getAdj();
```

```
00367
00368     for (auto &e : adjs)
00369     {
00370         auto w = e.getDest();
00371
00372         if (!w->isVisited())
00373         {
00374             dfsVisit(w, res);
00375         }
00376     }
00377
00378     return res;
00379 }
```

4.13.1.9 dfsVisit() [2/2]

```
std::vector< std::string > dfsVisit (
            Vertex< Airport > * v,
            std::vector< std::string > & res,
            int stop )
```

Search (depth-first) to visit vertices and collect connected countries withing a number of stops.

Parameters

airports	The graph of airports.
v	The vertex to start the search
stop	The number of stops.

Returns

vector<string> The vector with the countries.

Definition at line 389 of file dbairport.cpp.

```
00391 {
00392     if (stop == 0)
00393         return res;
00394
00395     // std::cout « "Saindo de " « v->getInfo().getCode() « std::endl;
00396
00397     v->setVisited(true);
00398     res.push_back(v->getInfo().getCountry());
00399     auto adjs = v->getAdj();
00400
00401     for (auto &e : adjs)
00402     {
00403         auto w = e.getDest();
00404
00405         if (!w->isVisited())
00406         {
00407             // std::cout « "\t Eu vou para: " « w->getInfo().getCode() «
00408             // std::endl;
00409             dfsVisit(w, res, (stop - 1));
00410         }
00411     }
00412     return res;
00413 }
```

4.13.1.10 distanceEarth()

```
double distanceEarth (
            double latOrigin,
            double longOrigin,
            double latDest,
            double longDest )
```

Function to calculate the distance between two points on the Earth Complexity: O(1)

Parameters

latOrigin	The latitude of the origin point.
longOrigin	The longitude of the origin point.
latDest	The latitude of the destination point.
longDest	The longitude of the destination point.

Returns

double The distance between the two points.

Definition at line 735 of file dbairport.cpp.

```
00737 {
00738     latOrigin = toRadians(latOrigin);
00739     longOrigin = toRadians(longOrigin);
00740     latDest = toRadians(latDest);
00741     longDest = toRadians(longDest);
00742
00743     double dlong = longDest - longOrigin;
00744     double dlat = latDest - latOrigin;
00745
00746     double ans = pow(sin(dlat / 2), 2) +
00747                  cos(latOrigin) * cos(latDest) * pow(sin(dlong / 2), 2);
00748
00749     ans = 2 * asin(sqrt(ans));
00750
00751     double R = 6371;
00752
00753     ans = ans * R;
00754
00755     return ans;
00756 }
```

4.13.1.11 findAirports() [1/2]

```
vector< Vertex< Airport > * > findAirports (
            Graph< Airport > & airports,
            double lat,
            double lon,
            int distMax )
```

Function to find all airports around a point withing a distance Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph representing airports and available flights.
lat	The latitude of the point.
lon	The longitude of the point.
distMax	The maximum distance.

Returns

vector<Vertex<Airport>*> A vector of airports.

Definition at line 770 of file dbairport.cpp.

```
00772 {
00773     vector<Vertex<Airport> *> vec;
```

```
00774     for (auto v : airports.getVertexSet())
00775     {
00776         double lat2 = v->getInfo().getLatitude();
00777         double lon2 = v->getInfo().getLongitude();
00778 
00779         double dist = distanceEarth(lat, lon, lat2, lon2);
00780 
00781         if (dist <= distMax)
00782         {
00783             vec.push_back(v);
00784         }
00785     }
00786     return vec;
00787 }
```

4.13.1.12 findAirports() [2/2]

```
vector< Vertex< Airport > * > findAirports (
            Graph< Airport > & airports,
            string country,
            string city )
```

Function to find all airports in a city Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph representing airports and available flights.
country	The country of the airport.
city	The city of the airport.

Returns

vector<Vertex<Airport>*> A vector of airports.

Definition at line 671 of file dbairport.cpp.

```
00673 {
00674     vector<Vertex<Airport> *> vec;
00675     for (auto v : airports.getVertexSet())
00676     {
00677         if (v->getInfo().getCountry() == country &&
00678             v->getInfo().getCity() == city)
00679         {
00680             vec.push_back(v);
00681         }
00682     }
00683     return vec;
00684 }
```

4.13.1.13 findArticulationPoints()

```
void findArticulationPoints (
            Graph< Airport > & airports )
```

Function to find and show articulation points in the graph. Points (airports) that, if removed, make certain regions of the map inaccessible Complexity: O(n^2), where n is the number of airports in the graph.

Parameters

airports	The graph representing airports and available flights.

Returns

void

Definition at line 1066 of file dbairport.cpp.

```
01067 {
01068     int i = 0;
01069     int total = 0;
01070     int connected = connectedComponents(airports);
01071     for (auto v : airports.getVertexSet())
01072     {
01073         // std::cout « " Analisando o vertice: " « v->getInfo().getCode() «
01074         // std::endl;
01075         resetVisited(airports);
01076         int component = 0;
01077         for (auto w : airports.getVertexSet())
01078         {
01079             // std::cout « "\tAnalisando o vertice: " « w->getInfo().getCode()«
01080             // std::endl;
01081             if (!w->isVisited() && w->getInfo().getCode() != v->getInfo().getCode())
01082             {
01083                 // std::cout « "\t\tChamando o DFS component++" « std::endl;
01084                 dfsArtc(v, w);
01085                 component++;
01086             }
01087         }
01088         if (component > connected)
01089         {
01090             std::cout « v->getInfo().getCode() « " ";
01091             i++;
01092             total++;
01093             if (i == 10)
01094             {
01095                 std::cout « std::endl;
01096                 i = 0;
01097             }
01098         }
01099     }
01100     std::cout « "\nTotal: " « total « std::endl;
01101 }
```

4.13.1.14 findBestFlights() [1/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            double latOrigin,
            double longOrigin,
            double latDest,
            double longDest,
            int distMax,
            vector< string > & airplanes )
```

Function to find the best flights between two points on Earth Point to Point This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
latOrigin	The latitude of the origin point.
longOrigin	The longitude of the origin point.
latDest	The latitude of the destination point.
longDest	The longitude of the destination point.
distMax	The maximum distance.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 802 of file dbairport.cpp.

```
00805 {
00806     vector<Vertex<Airport> *> src =
00807         findAirports(airports, latOrigin, longOrigin, distMax);
00808     vector<Vertex<Airport> *> dest =
00809         findAirports(airports, latDest, longDest, distMax);
00810     vector<vector<Flight>> paths;
00811
00812     showPath(airports, src, dest, paths, airplanes);
00813 }
```

4.13.1.15 findBestFlights() [2/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string airport,
            double lat,
            double lon,
            int distMax,
            int type,
            vector< string > & airplanes )
```

Function to find best flights between an airport and a point on Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p∗(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.

Parameters

airports	The graph representing airports and available flights.
airport	The airport.
lat	The latitude of the point.
lon	The longitude of the point.
distMax	The maximum distance.
type	The type of search (0: Point to Airport, 1: Airport to Point).
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 828 of file dbairport.cpp.

```
00831 {
00832     vector<Vertex<Airport> *> vec;
00833     vector<vector<Flight>> paths;
00834     vec = findAirports(airports, lat, lon, distMax);
00835
00836     if (type == 0)
00837     {
00838         for (auto v : vec)
00839         {
00840             std::cout « "Source: " « v->getInfo().getCode() « std::endl;
00841             resetVisited(airports);
00842             paths = bfsPath(v, airport, airplanes);
00843             showPath(paths);
00844         }
00845     }
00846     else if (type == 1)
00847     {
```

```
00848         for (auto v : vec)
00849         {
00850             std::cout « "Source: " « airport « std::endl;
00851             std::string tgt = v->getInfo().getCode();
00852             resetVisited(airports);
00853             paths = bfsPath(airports.findVertex(Airport(airport)), tgt, airplanes);
00854             showPath(paths);
00855         }
00856     }
00857 }
```

4.13.1.16 findBestFlights() [3/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string country,
            string city,
            double lat,
            double lon,
            int distMax,
            int type,
            vector< string > & airplanes )
```

Function to find best flights between a city and a point on the Earth This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(p∗(n + e)), where n is the number of airports, e is the number of flights in the graph and p is the number of source and destination airports.

Parameters

airports	The graph representing airports and available flights.
country	The country of the source airport.
city	The city of the source airport.
lat	The latitude of the point.
lon	The longitude of the point.
distMax	The maximum distance.
type	The type of search (0: City to Point, 1: Point to City).
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 878 of file dbairport.cpp.

```
00881 {
00882     vector<Vertex<Airport> *> vec_point;
00883     vector<Vertex<Airport> *> vec_city;
00884     vector<vector<Flight>> paths;
00885
00886     vec_point = findAirports(airports, lat, lon, distMax);
00887     vec_city = findAirports(airports, country, city);
00888
00889     if (type == 0)
00890     {
00891         for (auto c : vec_city)
00892         {
00893             std::cout « "Source: " « c->getInfo().getCode() « std::endl;
00894             for (auto p : vec_point)
00895             {
00896                 std::cout « "Destination: " « p->getInfo().getCode() « std::endl;
00897                 resetVisited(airports);
00898                 std::string tgt = p->getInfo().getCode();
00899                 paths = bfsPath(c, tgt, airplanes);
```

```
00900                 showPath(paths);
00901             }
00902         }
00903     }
00904     else if (type == 1)
00905     {
00906         for (auto p : vec_point)
00907         {
00908             std::cout « "Source: " « p->getInfo().getCode() « std::endl;
00909             for (auto c : vec_city)
00910             {
00911                 std::cout « "Destination: " « c->getInfo().getCode() « std::endl;
00912                 resetVisited(airports);
00913                 std::string tgt = c->getInfo().getCode();
00914                 paths = bfsPath(p, tgt, airplanes);
00915                 showPath(paths);
00916             }
00917         }
00918     }
00919 }
```

4.13.1.17 findBestFlights() [4/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string country,
            string city,
            string airport,
            int type,
            vector< string > & airplanes )
```

Finds the best flights between a city and an aiport (vice-verse) This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
country	The country of the source airport.
city	The city of the source airport.
airport	The destination airport.
type	The type of search (0: City to Airport, 1: Airport to City).
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 522 of file dbairport.cpp.

```
00524 {
00525     vector<Vertex<Airport> *> vec;
00526     vector<vector<Flight>> paths;
00527     vec = findAirports(airports, country, city);
00528
00529     if (type == 0)
00530     {
00531         for (auto v : vec)
00532         {
00533             // std::cout « "Source: " « v->getInfo().getCode() « std::endl;
00534             resetVisited(airports);
00535             paths = bfsPath(v, airport, airplanes);
00536             showPath(paths);
00537         }
00538     }
00539     else if (type == 1)
```

```
00540     {
00541         for (auto v : vec)
00542         {
00543             // std::cout « "Source: " « airport « std::endl;
00544             std::string tgt = v->getInfo().getCode();
00545             resetVisited(airports);
00546             paths = bfsPath(airports.findVertex(Airport(airport)), tgt, airplanes);
00547             showPath(paths);
00548         }
00549     }
00550 }
```

4.13.1.18 findBestFlights() [5/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string countrySrc,
            string citySrc,
            string countryDest,
            string cityDest,
            vector< string > & airplanes )
```

Function to find best flights between two cities This function uses breadth-first search (BFS) to find paths between a source and destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
countrySrc	The country of the source airport.
citySrc	The city of the source airport.
countryDest	The country of the destination airport.
cityDest	The city of the destination airport.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 698 of file dbairport.cpp.

```
00701 {
00702     vector<Vertex<Airport> *> src;
00703     vector<Vertex<Airport> *> dest;
00704     vector<vector<Flight>> paths;
00705
00706     src = findAirports(airports, countrySrc, citySrc);
00707     dest = findAirports(airports, countryDest, cityDest);
00708
00709     showPath(airports, src, dest, paths, airplanes);
00710 }
```

4.13.1.19 findBestFlights() [6/6]

```
void findBestFlights (
            Graph< Airport > & airports,
            string src,
            string dest,
            vector< string > & airplanes )
```

Function to find the best flights from a source airport to a destination airport Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
src	The source airport.
dest	The destination airport.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 494 of file dbairport.cpp.

```
00496 {
00497     resetVisited(airports);
00498
00499     auto s = airports.findVertex(Airport(src));
00500     auto d = airports.findVertex(Airport(dest));
00501
00502     if (s == nullptr || d == nullptr)
00503         return;
00504
00505     vector<vector<Flight>> paths;
00506     paths = bfsPath(s, dest, airplanes);
00507     showPath(paths);
00508 }
```

4.13.1.20 getPath()

```
void getPath (
            string current,
            vector< Flight > & path,
            unordered_map< string, vector< Flight > > & prev,
            vector< vector< Flight > > & paths,
            string startCode,
            string airline )
```

Function to find a path from the current airport to start airport Complexity: O(n), where n is the number of airports in the graph.

Parameters

current	The current airport.
path	The current path.
prev	A map with the previous airports.
paths	A vector of vectors of flights representing the paths.
startCode	The code of the start airport.
airline	The airline of the flight.

Returns

void

Definition at line 932 of file dbairport.cpp.

```
00935 {
00936     path.push_back({current, airline});
00937
```

```
00938     if (current == startCode)
00939     {
00940         vector<Flight> validPath = path;
00941         reverse(validPath.begin(), validPath.end());
00942         paths.push_back(validPath);
00943     }
00944     else
00945     {
00946         for (auto &prevVertex : prev[current])
00947         {
00948             getPath(prevVertex.code, path, prev, paths, startCode,
00949                     prevVertex.airline);
00950         }
00951     }
00952
00953     path.pop_back();
00954 }
```

4.13.1.21 quantityAirlinesCountry()

```
int quantityAirlinesCountry (
            unordered_map< string, Airline > airlines,
            std::string country )
```

Function to calculate the quantity of airlines for a given country Complexity: O(n), where n is the number of airlines in the graph.

Parameters

airlines	The airlines hashtable.
country	The country to be searched.

Returns

int The quantity of airlines in the country.

Definition at line 90 of file dbairport.cpp.

```
00091 {
00092     int count = 0;
00093     for (auto v : airlines)
00094     {
00095         if (v.second.getCountry() == country)
00096         {
00097             count++;
00098         }
00099     }
00100     return count;
00101 }
```

4.13.1.22 quantityAirports()

```
int quantityAirports (
            Graph< Airport > airports )
```

Function to get quantity of airports in the graph Complexity: O(1)

Parameters

airports	The graph of airports.

Returns

int The quantity of airports in the graph.

Definition at line 11 of file dbairport.cpp.

```
00012 {
00013     return airports.getNumVertex();
00014 }
```

4.13.1.23 quantityAirportsCity()

```
int quantityAirportsCity (
            Graph< Airport > airports,
            std::string city )
```

Function to get quantity of airports in the graph by city Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
city	The city to be searched.

Returns

int The quantity of airports in the city.

Definition at line 44 of file dbairport.cpp.

```
00045 {
00046     int count = 0;
00047     for (auto v : airports.getVertexSet())
00048     {
00049         Airport a = v->getInfo();
00050         if (a.getCity() == city)
00051         {
00052             count++;
00053         }
00054     }
00055     return count;
00056 }
```

4.13.1.24 quantityAirportsCountry()

```
int quantityAirportsCountry (
            Graph< Airport > airports,
            std::string country )
```

Function to get quantity of airports in the graph by country Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
country	The country to be searched.

Returns

int The quantity of airports in the country.

Definition at line 23 of file dbairport.cpp.

```
00024 {
00025     int count = 0;
00026     for (auto v : airports.getVertexSet())
00027     {
00028         Airport a = v->getInfo();
00029         if (a.getCountry() == country)
00030         {
00031             count++;
00032         }
00033     }
00034     return count;
00035 }
```

4.13.1.25 quantityCitiesCountry()

```
int quantityCitiesCountry (
            Graph< Airport > airports,
            std::string country )
```

Function to count the number of unique cities in the graph for a given country Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
country	The country to be searched.

Returns

int The quantity of unique cities in the country.

Definition at line 65 of file dbairport.cpp.

```
00066 {
00067     std::unordered_set<std::string> uniqueCities;
00068     for (auto v : airports.getVertexSet())
00069     {
00070         Airport a = v->getInfo();
00071         if (a.getCountry() == country)
00072         {
00073             std::string city = a.getCity();
00074             uniqueCities.insert(city);
00075         }
00076     }
00077     return uniqueCities.size();
00078 }
```

4.13.1.26 quantityDestinationLimitedStop()

```
int quantityDestinationLimitedStop (
            Graph< Airport > airports,
            std::string airport,
            int stop )
```

Function to calculate the number of different countries it is possible to go to from an airport with a given number of stops Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.
airport	The airport to be searched.
stop	The number of stops.

Returns

int The quantity of countries.

Definition at line 241 of file dbairport.cpp.

```
00243 {
00244     std::set<std::string> countries;
00245     std::vector<std::string> res;
00246     resetVisited(airports);
00247     auto s = airports.findVertex(Airport(airport));
00248     if (s == nullptr)
00249         return 0;
00250     res = dfsVisit(s, res, (stop + 1));
00251     for (auto c : res)
00252     {
00253         countries.insert(c);
00254     }
00255     return countries.size();
00256 }
```

4.13.1.27 quantityDestinationMax()

```
int quantityDestinationMax (
            Graph< Airport > airports )
```

Function that calculates the maximum possible trip, with the greatest number of stops and print the starting airport and the destination airport Complexity: O(n ∗ (n + e)), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.

Returns

int The number of stops on the longest trip

Definition at line 265 of file dbairport.cpp.

```
00266 {
00267     vector<string> path, maxPath;
00268
00269     for (auto v : airports.getVertexSet())
00270     {
00271         // std::cout « "Consultando... " « v->getInfo().getCode() « std::endl;
00272         resetVisited(airports);
00273         path.clear();
00274         dfsMax(v, path, maxPath);
00275     }
00276
00277     std::cout « "\nStarting in: " « maxPath[0] « std::endl;
00278     // for (int i = 1; i < maxPath.size() - 1; i++)
00279     // {
00280     //     std::cout « maxPath[i] « " -> ";
00281     // }
00282     std::cout « "\nEnding in: " « maxPath[maxPath.size() - 1] « std::endl;
00283     return maxPath.size() - 1;
00284 }
```

4.13.1.28 quantityDestinationsAirport()

```
int quantityDestinationsAirport (
            Graph< Airport > airports,
            std::string airport )
```

Function to calculate the number of different countries it is possible to go directly to from an airport Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.
airport	The airport to be searched.

Returns

int The quantity of countries.

Definition at line 216 of file dbairport.cpp.

```
00217 {
00218     std::set<std::string> countries;
00219     std::vector<std::string> res;
00220     resetVisited(airports);
00221     auto s = airports.findVertex(Airport(airport));
00222     if (s == nullptr)
00223         return 0;
00224     Graph<Airport> g;
00225     res = dfsVisit(s, res);
00226     for (auto c : res)
00227     {
00228         countries.insert(c);
00229     }
00230     return countries.size();
00231 }
```

4.13.1.29 quantityFlights() [1/2]

```
int quantityFlights (
            Graph< Airport > airports )
```

Function to calculate the quatity of flights in the graph Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.

Returns

int The quantity of flights (edges) in the graph.

Definition at line 111 of file dbairport.cpp.

```
00112 {
00113     int count = 0;
00114     for (auto v : airports.getVertexSet())
00115     {
00116         count += v->getAdj().size();
00117     }
00118     return count;
00119 }
```

4.13.1.30 quantityFlights() [2/2]

```
std::pair< int, int > quantityFlights (
            Graph< Airport > airports,
            std::string code )
```

Function to get quantity of flights in the graph and unique airlines for an Airport Complexity: O(n), where n is the number of flights in the graph.

Parameters

airports	The graph of airports.
code	The airport to be searched.

Returns

pair<int, int> The quantity of flights and unique airlines.

Definition at line 338 of file dbairport.cpp.

```
00339 {
00340     int count = 0;
00341     std::set<std::string> airlines;
00342 
00343     auto s = airports.findVertex(Airport(code));
00344     if (s != nullptr)
00345     {
00346         for (auto v : s->getAdj())
00347         {
00348             count++;
00349             airlines.insert(v.getRoute());
00350         }
00351     }
00352     return std::pair<int, int>(count, airlines.size());
00353 }
```

4.13.1.31 quantityFlightsAirline()

```
int quantityFlightsAirline (
            Graph< Airport > airports,
            std::string airline )
```

Function to calculate the quatity of flights in a given airline Complexity: O(n + e), where n is the number of airports in the graph and e is the number of flights.

Parameters

airports	The graph of airports.
airline	The airline to be searched.

Returns

int The quantity of flights (edges) in the airline.

Definition at line 190 of file dbairport.cpp.

```
00191 {
00192     int count = 0;
00193     for (auto v : airports.getVertexSet())
```

```
00194     {
00195         for (auto e : v->getAdj())
00196         {
00197             if (e.getRoute() == airline)
00198             {
00199                 count++;
00200             }
00201         }
00202     }
00203
00204     return count;
00205 }
```

4.13.1.32 quantityFlightsAirport()

```
int quantityFlightsAirport (
            Graph< Airport > airports,
            std::string airport )
```

Function to calculate the quatity of flights in a given airport Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
airport	The airport to be searched.

Returns

int The quantity of flights (edges) in the airport.

Definition at line 128 of file dbairport.cpp.

```
00129 {
00130     int count = 0;
00131     for (auto v : airports.getVertexSet())
00132     {
00133         if (v->getInfo().getCode() == airport)
00134         {
00135             count += v->getAdj().size();
00136         }
00137     }
00138     return count;
00139 }
```

4.13.1.33 quantityFlightsCity()

```
int quantityFlightsCity (
            Graph< Airport > airports,
            std::string city )
```

Function to calculate the quatity of flights in a given city Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
city	The city to be searched.

Returns

int The quantity of flights (edges) in the city.

Definition at line 169 of file dbairport.cpp.

```
00170 {
00171     int count = 0;
00172     for (auto v : airports.getVertexSet())
00173     {
00174         if (v->getInfo().getCity() == city)
00175         {
00176             count += v->getAdj().size();
00177         }
00178     }
00179     return count;
00180 }
```

4.13.1.34 quantityFlightsCountry()

```
int quantityFlightsCountry (
            Graph< Airport > airports,
            std::string country )
```

Function to calculate the quatity of flights in a given country Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.
country	The country to be searched.

Returns

int The quantity of flights (edges) in the country.

Definition at line 148 of file dbairport.cpp.

```
00149 {
00150     int count = 0;
00151     for (auto v : airports.getVertexSet())
00152     {
00153         if (v->getInfo().getCountry() == country)
00154         {
00155             count += v->getAdj().size();
00156         }
00157     }
00158     return count;
00159 }
```

4.13.1.35 rankingAirports()

```
void rankingAirports (
            Graph< Airport > airports,
            int arg )
```

Function to rank airport based of the sum of in-degrees and out-degrees Complexity: O(n $*$ log(n) + n $*$ e), where n is the number of airports and e is the number of flights in the graph (assuming sorting has a time complexity of O(n $*$ log(n)))

Parameters

airports	The graph of airports.
arg	The number of airports to be shown.

Returns

void

Definition at line 460 of file dbairport.cpp.

```
00461 {
00462     std::vector<Ranking> vec;
00463     calculateIndegree(airports);
00464
00465     for (auto v : airports.getVertexSet())
00466     {
00467         int total = v->getIndegree() + v->getAdj().size();
00468         Ranking rank = {v->getInfo().getCode(), total};
00469         vec.push_back(rank);
00470     }
00471     std::sort(vec.begin(), vec.end(), comparator);
00472     int i = 0;
00473     for (auto v : vec)
00474     {
00475         if (i < arg)
00476         {
00477             std::cout << "Code: " << v.code << " / ";
00478             std::cout << "Name: " << airportsHash.find(v.code)->second.getName() << " / ";
00479             std::cout << "Total: " << v.count << std::endl;
00480             i++;
00481         }
00482     }
00483 }
```

4.13.1.36 resetVisited()

```
void resetVisited (
            Graph< Airport > & airports )
```

Function to reset the visited status of all vertices in the graph Complexity: O(n), where n is the number of airports in the graph.

Parameters

airports	The graph of airports.

Returns

void

Definition at line 421 of file dbairport.cpp.

```
00422 {
00423     for (auto v : airports.getVertexSet())
00424     {
00425         v->setVisited(false);
00426     }
00427 }
```

4.13.1.37 showPath() [1/2]

```
void showPath (
            Graph< Airport > & airports,
```

```
        vector< Vertex< Airport > * > source,
        vector< Vertex< Airport > * > dest,
        vector< vector< Flight > > paths,
        vector< string > & airplanes )
```

Function to show a multi-path between multiple source and destination airports Complexity: O(n + e), where n is the number of airports and e is the number of flights in the graph.

Parameters

airports	The graph representing airports and available flights.
source	A vector of source airports.
dest	A vector of destination airports.
paths	A vector of vectors of flights representing the paths.
airplanes	A vector of strings representing allowed aircraft (can be empty to consider all).

Returns

void

Definition at line 563 of file dbairport.cpp.

```
00566 {
00567
00568     vector<vector<Flight>> flights;
00569
00570     for (auto s : source)
00571     {
00572         // std::cout « "Source: " « s->getInfo().getCode() « std::endl;
00573         for (auto d : dest)
00574         {
00575             // std::cout « "Destination: " « d->getInfo().getCode() «
00576             // std::endl;
00577             std::string tgt = d->getInfo().getCode();
00578             resetVisited(airports);
00579             flights = bfsPath(s, tgt, airplanes);
00580             paths.insert(paths.end(), flights.begin(), flights.end());
00581         }
00582     }
00583
00584     if (paths.empty())
00585     {
00586         std::cout « "--------------------------------------------------"
00587                   « std::endl;
00588         std::cout « "Sorry, but there is no result with those inputs."
00589                   « std::endl;
00590         return;
00591     }
00592     else
00593     {
00594
00595         std::sort(paths.begin(), paths.end(), comparatorPath);
00596         const int min = paths[0].size();
00597
00598         for (auto &p : paths)
00599         {
00600             if (p.size() > min)
00601             {
00602                 continue;
00603             }
00604             std::cout « "--------------------------------------------------"
00605                       « std::endl;
00606             for (auto &f : p)
00607             {
00608                 if (!f.airline.empty())
00609                 {
00610                     std::cout « f.code « " -(" « f.airline « ")"
00611                               « "-> ";
00612                 }
00613                 else
00614                 {
00615                     std::cout « f.code;
00616                 }
00617             }
```

```
00618             std::cout « std::endl;
00619         }
00620     }
00621 }
```

4.13.1.38 showPath() [2/2]

```
void showPath (
            vector< vector< Flight > > paths )
```

Function to show a set of paths Complexity: O(n), where n is the total number of flights in all paths.

Parameters

paths	A vector of vectors of flights representing the paths.

Returns

void

Definition at line 632 of file dbairport.cpp.

```
00633 {
00634     if (paths.empty())
00635     {
00636         std::cout « "--------------------------------------------------"
00637                   « std::endl;
00638         std::cout « "Sorry, but there is no result with those inputs."
00639                   « std::endl;
00640         return;
00641     }
00642
00643     for (auto path : paths)
00644     {
00645         std::cout « "--------------------------------------------------"
00646                   « std::endl;
00647         for (auto p : path)
00648         {
00649             if (!p.airline.empty())
00650             {
00651                 std::cout « p.code « " -(" « p.airline « ")"
00652                           « "-> ";
00653             }
00654             else
00655             {
00656                 std::cout « p.code;
00657             }
00658         }
00659         std::cout « std::endl;
00660     }
00661 }
```

4.13.1.39 toRadians()

```
double toRadians (
            const double degree )
```

Function to convert Degrees in radians Complexity: O(1)

Parameters

degree	The degree to be converted.

Returns

double The converted degree.

Definition at line 718 of file dbairport.cpp.

```
00718 { return (degree * M_PI / 180); }
```

4.14 dbairport.h

Go to the documentation of this file.

```
00001 #ifndef DBAIRPORT_H
00002 #define DBAIRPORT_H
00003
00004 #include "../classes/Airline.h"
00005 #include "../classes/Airport.h"
00006 #include "../classes/Graph.h"
00007 #include "read.h"
00008 #include <algorithm>
00009 #include <cmath>
00010 #include <fstream>
00011 #include <limits.h>
00012 #include <map>
00013 #include <set>
00014 #include <unordered_map>
00015 #include <unordered_set>
00016
00017 struct Flight
00018 {
00019   std::string code;
00020   std::string airline;
00021 };
00022
00023 // Quantity Airports Functions
00024 int quantityAirports(Graph<Airport> airports);
00025 int quantityAirportsCountry(Graph<Airport> airports, std::string country);
00026 int quantityAirportsCity(Graph<Airport> airports, std::string city);
00027
00028 // Quantity Cities Functions
00029 int quantityCitiesCountry(Graph<Airport> airports, std::string country);
00030
00031 // Quantity Airlines Functions
00032 int quantityAirlinesCountry(unordered_map<string, Airline> airlines, std::string country);
00033
00034 // Quantity Flights Functions
00035 int quantityFlights(Graph<Airport> airports);
00036 int quantityFlightsAirport(Graph<Airport> airports, std::string airport);
00037 int quantityFlightsCountry(Graph<Airport> airports, std::string country);
00038 int quantityFlightsCity(Graph<Airport> airports, std::string city);
00039 int quantityFlightsAirline(Graph<Airport> airports, std::string airline);
00040
00041 // Quantity Destination Functions
00042 int quantityDestinationsAirport(Graph<Airport> airports, std::string airport);
00043 int quantityDestinationLimitedStop(Graph<Airport> airports,
00044                                    std::string airport, int stop);
00045 int quantityDestinationMax(Graph<Airport> airports);
00046
00047 // Other functions
00048 std::pair<int, int> quantityFlights(Graph<Airport> airports, std::string code);
00049 std::vector<std::string> dfsVisit(Vertex<Airport> *v,
00050                                   std::vector<std::string> &res);
00051 std::vector<std::string> dfsVisit(Vertex<Airport> *v,
00052                                   std::vector<std::string> &res, int stop);
00053 void resetVisited(Graph<Airport> &airports);
00054 void rankingAirports(Graph<Airport> airports, int arg);
00055 void calculateIndegree(Graph<Airport> &airports);
00056
00057 // Best Flights Functions
00058 void findBestFlights(Graph<Airport> &airports, string src, string dest,
00059                      vector<string> &airplanes);
00060 void findBestFlights(Graph<Airport> &airports, string countrySrc,
00061                      string citySrc, string countryDest, string cityDest,
00062                      vector<string> &airplanes);
00063 void findBestFlights(Graph<Airport> &airports, double latOrigin,
00064                      double longOrigin, double latDest, double longDest,
00065                      int distMax, vector<string> &airplanes);
00066 void findBestFlights(Graph<Airport> &airports, string country, string city,
00067                      string airport, int type, vector<string> &airplanes);
00068 void findBestFlights(Graph<Airport> &airports, string country, string city,
00069                      double lat, double lon, int distMax, int type,
```

```
00070                     vector<string> &airplanes);
00071 void findBestFlights(Graph<Airport> &airports, string airport, double lat,
00072                     double lon, int distMax, int type,
00073                     vector<string> &airplanes);
00074 //------------------------------------------------------------
00075 void showPath(vector<vector<Flight» paths);
00076 void showPath(Graph<Airport> &airports, vector<Vertex<Airport> *> source,
00077               vector<Vertex<Airport> *> dest, vector<vector<Flight» paths,
00078               vector<string> &airplanes);
00079 //------------------------------------------------------------
00080 vector<Vertex<Airport> *> findAirports(Graph<Airport> &airports, string country,
00081                                        string city);
00082 vector<Vertex<Airport> *> findAirports(Graph<Airport> &airports, double lat,
00083                                        double lon, int distMax);
00084 //------------------------------------------------------------
00085 vector<vector<Flight» bfsPath(Vertex<Airport> *v, string &tgt,
00086                               vector<string> &airplanes);
00087 void getPath(string current, vector<Flight> &path,
00088              unordered_map<string, vector<Flight» &prev,
00089              vector<vector<Flight» &paths, string startCode, string airline);
00090 void dfsArtc(Vertex<Airport> *v, Vertex<Airport> *w);
00091 void findArticulationPoints(Graph<Airport> &airports);
00092 bool comparatorPath(const vector<Flight> a, const vector<Flight> b);
00093 //------------------------------------------------------------
00094 void dfsConnectedComponents(Graph<Airport> &airports, Vertex<Airport> *v);
00095 int connectedComponents(Graph<Airport> &airports);
00096 void dfsMax(Vertex<Airport> *v, std::vector<std::string> &path, std::vector<std::string> &maxPath);
00097 double toRadians(const double degree);
00098 double distanceEarth(double latOrigin, double longOrigin, double latDest,
00099                     double longDest);
00100
00101 #endif // DBAIRPORT_H
```

4.15 src/database/read.cpp File Reference

```
#include "read.h"
```

Functions

- Graph< Airport > readAirports (std::string folder)

  Function to read Airport data from CSV file and create graph of airports Complexity: O(n), where n is the number of airports in the csv file.
- Graph< Airport > readFlights (std::string folder)

  Function to read Flight data from CSV file and add edges to graph of airports Complexity: O(n + v), where n is the number of airports and v is the number of flights in the csv file.
- unordered_map< string, Airline > readAirlines (std::string folder)

  Function to read Airline data from CSV file and create a graph of airlines Complexity: O(n), where n is the number of airlines in the CSV file.

Variables

- unordered_map< string, Airport > airportsHash
- unordered_set< string > citiesHash
- unordered_set< string > countriesHash

4.15.1 Function Documentation

4.15.1.1 readAirlines()

```
unordered_map< string, Airline > readAirlines (
            std::string folder )
```

Function to read Airline data from CSV file and create a graph of airlines Complexity: O(n), where n is the number of airlines in the CSV file.

Parameters

folder	The folder where the CSV file is located.

Returns

Definition at line 122 of file read.cpp.

```
00123 {
00124   // Initialize Graph
00125   unordered_map<string, Airline> airlines;
00126
00127   // Open csv file
00128   std::ifstream file("../dataset/" + folder + "/airlines.csv", ios::in);
00129   if (!file.is_open())
00130   {
00131     std::cout « "Error opening file" « std::endl;
00132     return airlines;
00133   }
00134
00135   // Read and ignore header line
00136   std::string line;
00137   if (!std::getline(file, line))
00138   {
00139     std::cout « "Error reading header line" « std::endl;
00140     file.close();
00141     return airlines;
00142   }
00143
00144   // Read each line and create an Airline object
00145   while (std::getline(file, line))
00146   {
00147     std::istringstream iss(line);
00148     std::string code, name, callsign, country;
00149
00150     std::getline(iss, code, ',');
00151     std::getline(iss, name, ',');
00152     std::getline(iss, callsign, ',');
00153     std::getline(iss, country, ',');
00154
00155     Airline airline(code, name, callsign, country);
00156     // airline.display();
00157     airlines.insert({code, airline});
00158   }
00159
00160   // Close file and return the graph of airlines
00161   file.close();
00162   return airlines;
00163 }
```

4.15.1.2 readAirports()

```
Graph< Airport > readAirports (
            std::string folder )
```

Function to read Airport data from CSV file and create graph of airports Complexity: O(n), where n is the number of airports in the csv file.

Parameters

folder	The folder where the CSV file is located.

Returns

Graph<Airport> The graph of airports.

Definition at line 12 of file read.cpp.

```
00013 {
00014   // Inicialiate Graph
00015   Graph<Airport> airports = Graph<Airport>();
00016
00017   // Open csv file
00018   std::ifstream file("../dataset/" + folder + "/airports.csv", ios::in);
00019   if (!file.is_open())
00020   {
00021     std::cout « "Error opening file" « std::endl;
00022     return airports;
00023   }
00024
00025   // Read and ignore header line
00026   std::string line;
00027   if (!std::getline(file, line))
00028   {
00029     std::cout « "Error reading header line" « std::endl;
00030     file.close();
00031     return airports;
00032   }
00033
00034   // Read each line and create an Airport object
00035   while (std::getline(file, line))
00036   {
00037     std::istringstream iss(line);
00038     std::string code, name, country, city;
00039     double latitude, longitude;
00040
00041     std::getline(iss, code, ',');
00042     std::getline(iss, name, ',');
00043     std::getline(iss, city, ',');
00044     std::getline(iss, country, ',');
00045
00046     iss » latitude;
00047     iss.ignore();
00048     iss » longitude;
00049
00050     Airport airport(code, name, country, city, latitude, longitude);
00051
00052     airportsHash.insert({code, airport});
00053     citiesHash.insert(city);
00054     countriesHash.insert(country);
00055
00056     // airport.display();
00057     airports.addVertex(airport);
00058   }
00059
00060   // Close file and return graph of airports
00061   file.close();
00062   return airports;
00063 }
```

4.15.1.3 readFlights()

```
Graph< Airport > readFlights (
            std::string folder )
```

Function to read Flight data from CSV file and add edges to graph of airports Complexity: O(n + v), where n is the number of airports and v is the number of flights in the csv file.

Parameters

folder	The folder where the CSV file is located.

Returns

Graph<Airport> The graph of airports with flights.

Definition at line 72 of file read.cpp.

```
00073 {
00074   // Read airports
```

```
00075   Graph<Airport> airports = readAirports(folder);
00076
00077   // Open csv file
00078   std::ifstream file("../dataset/" + folder + "/flights.csv", ios::in);
00079   if (!file.is_open())
00080   {
00081     std::cout « "Error opening flights file" « std::endl;
00082     return airports;
00083   }
00084
00085   // Read and ignore header line
00086   std::string line;
00087   if (!std::getline(file, line))
00088   {
00089     std::cout « "Error reading flights header line" « std::endl;
00090     file.close();
00091     return airports;
00092   }
00093
00094   // Read each line and add edges to graph
00095   while (std::getline(file, line))
00096   {
00097     std::istringstream iss(line);
00098     std::string source, target, airline;
00099
00100     std::getline(iss, source, ',');
00101     std::getline(iss, target, ',');
00102     std::getline(iss, airline, ',');
00103
00104     Airport src = Airport(source);
00105     Airport tgt = Airport(target);
00106
00107     airports.addEdge(src, tgt, 0, airline);
00108   }
00109
00110   // Close file and return the updated graph of airports with flights
00111   file.close();
00112   return airports;
00113 }
```

4.15.2 Variable Documentation

4.15.2.1 airportsHash

```
unordered_map<string, Airport> airportsHash
```

Definition at line 3 of file read.cpp.

4.15.2.2 citiesHash

```
unordered_set<string> citiesHash
```

Definition at line 4 of file read.cpp.

4.15.2.3 countriesHash

```
unordered_set<string> countriesHash
```

Definition at line 5 of file read.cpp.

4.16 read.cpp

Go to the documentation of this file.

```
00001 #include "read.h"
00002
00003 unordered_map<string, Airport> airportsHash;
00004 unordered_set<string> citiesHash;
00005 unordered_set<string> countriesHash;
00012 Graph<Airport> readAirports(std::string folder)
00013 {
00014   // Inicialiate Graph
00015   Graph<Airport> airports = Graph<Airport>();
00016
00017   // Open csv file
00018   std::ifstream file("../dataset/" + folder + "/airports.csv", ios::in);
00019   if (!file.is_open())
00020   {
00021     std::cout « "Error opening file" « std::endl;
00022     return airports;
00023   }
00024
00025   // Read and ignore header line
00026   std::string line;
00027   if (!std::getline(file, line))
00028   {
00029     std::cout « "Error reading header line" « std::endl;
00030     file.close();
00031     return airports;
00032   }
00033
00034   // Read each line and create an Airport object
00035   while (std::getline(file, line))
00036   {
00037     std::istringstream iss(line);
00038     std::string code, name, country, city;
00039     double latitude, longitude;
00040
00041     std::getline(iss, code, ',');
00042     std::getline(iss, name, ',');
00043     std::getline(iss, city, ',');
00044     std::getline(iss, country, ',');
00045
00046     iss » latitude;
00047     iss.ignore();
00048     iss » longitude;
00049
00050     Airport airport(code, name, country, city, latitude, longitude);
00051
00052     airportsHash.insert({code, airport});
00053     citiesHash.insert(city);
00054     countriesHash.insert(country);
00055
00056     // airport.display();
00057     airports.addVertex(airport);
00058   }
00059
00060   // Close file and return graph of airports
00061   file.close();
00062   return airports;
00063 }
00064
00072 Graph<Airport> readFlights(std::string folder)
00073 {
00074   // Read airports
00075   Graph<Airport> airports = readAirports(folder);
00076
00077   // Open csv file
00078   std::ifstream file("../dataset/" + folder + "/flights.csv", ios::in);
00079   if (!file.is_open())
00080   {
00081     std::cout « "Error opening flights file" « std::endl;
00082     return airports;
00083   }
00084
00085   // Read and ignore header line
00086   std::string line;
00087   if (!std::getline(file, line))
00088   {
00089     std::cout « "Error reading flights header line" « std::endl;
00090     file.close();
00091     return airports;
00092   }
00093
00094   // Read each line and add edges to graph
00095   while (std::getline(file, line))
```

```
00096   {
00097     std::istringstream iss(line);
00098     std::string source, target, airline;
00099
00100     std::getline(iss, source, ',');
00101     std::getline(iss, target, ',');
00102     std::getline(iss, airline, ',');
00103
00104     Airport src = Airport(source);
00105     Airport tgt = Airport(target);
00106
00107     airports.addEdge(src, tgt, 0, airline);
00108   }
00109
00110   // Close file and return the updated graph of airports with flights
00111   file.close();
00112   return airports;
00113 }
00114
00115 // Update later to tree hashtable
00122 unordered_map<string, Airline> readAirlines(std::string folder)
00123 {
00124   // Initialize Graph
00125   unordered_map<string, Airline> airlines;
00126
00127   // Open csv file
00128   std::ifstream file("../dataset/" + folder + "/airlines.csv", ios::in);
00129   if (!file.is_open())
00130   {
00131     std::cout « "Error opening file" « std::endl;
00132     return airlines;
00133   }
00134
00135   // Read and ignore header line
00136   std::string line;
00137   if (!std::getline(file, line))
00138   {
00139     std::cout « "Error reading header line" « std::endl;
00140     file.close();
00141     return airlines;
00142   }
00143
00144   // Read each line and create an Airline object
00145   while (std::getline(file, line))
00146   {
00147     std::istringstream iss(line);
00148     std::string code, name, callsign, country;
00149
00150     std::getline(iss, code, ',');
00151     std::getline(iss, name, ',');
00152     std::getline(iss, callsign, ',');
00153     std::getline(iss, country, ',');
00154
00155     Airline airline(code, name, callsign, country);
00156     // airline.display();
00157     airlines.insert({code, airline});
00158   }
00159
00160   // Close file and return the graph of airlines
00161   file.close();
00162   return airlines;
00163 }
```

4.17 src/database/read.h File Reference

```
#include "../classes/Airline.h"
#include "../classes/Airport.h"
#include "../classes/Graph.h"
#include <filesystem>
#include <fstream>
#include <iostream>
#include <sstream>
#include <unordered_map>
#include <unordered_set>
```

Functions

- unordered_map< string, Airline > readAirlines (std::string folder)

  Function to read Airline data from CSV file and create a graph of airlines Complexity: O(n), where n is the number of airlines in the CSV file.
- Graph< Airport > readAirports (std::string folder)

  Function to read Airport data from CSV file and create graph of airports Complexity: O(n), where n is the number of airports in the csv file.
- Graph< Airport > readFlights (std::string folder)

  Function to read Flight data from CSV file and add edges to graph of airports Complexity: O(n + v), where n is the number of airports and v is the number of flights in the csv file.

Variables

- unordered_map< string, Airport > airportsHash
- unordered_set< string > citiesHash
- unordered_set< string > countriesHash

4.17.1 Function Documentation

4.17.1.1 readAirlines()

```
unordered_map< string, Airline > readAirlines (
            std::string folder )
```

Function to read Airline data from CSV file and create a graph of airlines Complexity: O(n), where n is the number of airlines in the CSV file.

Parameters

folder	The folder where the CSV file is located.

Returns

Definition at line 122 of file read.cpp.

```
00123 {
00124   // Initialize Graph
00125   unordered_map<string, Airline> airlines;
00126
00127   // Open csv file
00128   std::ifstream file("../dataset/" + folder + "/airlines.csv", ios::in);
00129   if (!file.is_open())
00130   {
00131     std::cout « "Error opening file" « std::endl;
00132     return airlines;
00133   }
00134
00135   // Read and ignore header line
00136   std::string line;
00137   if (!std::getline(file, line))
00138   {
00139     std::cout « "Error reading header line" « std::endl;
00140     file.close();
00141     return airlines;
00142   }
00143
00144   // Read each line and create an Airline object
```

```
00145   while (std::getline(file, line))
00146   {
00147     std::istringstream iss(line);
00148     std::string code, name, callsign, country;
00149
00150     std::getline(iss, code, ',');
00151     std::getline(iss, name, ',');
00152     std::getline(iss, callsign, ',');
00153     std::getline(iss, country, ',');
00154
00155     Airline airline(code, name, callsign, country);
00156     // airline.display();
00157     airlines.insert({code, airline});
00158   }
00159
00160   // Close file and return the graph of airlines
00161   file.close();
00162   return airlines;
00163 }
```

4.17.1.2 readAirports()

```
Graph< Airport > readAirports (
            std::string folder )
```

Function to read Airport data from CSV file and create graph of airports Complexity: O(n), where n is the number of airports in the csv file.

Parameters

folder	The folder where the CSV file is located.

Returns

Graph<Airport> The graph of airports.

Definition at line 12 of file read.cpp.

```
00013 {
00014   // Inicialiate Graph
00015   Graph<Airport> airports = Graph<Airport>();
00016
00017   // Open csv file
00018   std::ifstream file("../dataset/" + folder + "/airports.csv", ios::in);
00019   if (!file.is_open())
00020   {
00021     std::cout « "Error opening file" « std::endl;
00022     return airports;
00023   }
00024
00025   // Read and ignore header line
00026   std::string line;
00027   if (!std::getline(file, line))
00028   {
00029     std::cout « "Error reading header line" « std::endl;
00030     file.close();
00031     return airports;
00032   }
00033
00034   // Read each line and create an Airport object
00035   while (std::getline(file, line))
00036   {
00037     std::istringstream iss(line);
00038     std::string code, name, country, city;
00039     double latitude, longitude;
00040
00041     std::getline(iss, code, ',');
00042     std::getline(iss, name, ',');
00043     std::getline(iss, city, ',');
00044     std::getline(iss, country, ',');
00045
00046     iss » latitude;
00047     iss.ignore();
```

```
00048     iss » longitude;
00049
00050     Airport airport(code, name, country, city, latitude, longitude);
00051
00052     airportsHash.insert({code, airport});
00053     citiesHash.insert(city);
00054     countriesHash.insert(country);
00055
00056     // airport.display();
00057     airports.addVertex(airport);
00058   }
00059
00060   // Close file and return graph of airports
00061   file.close();
00062   return airports;
00063 }
```

4.17.1.3 readFlights()

```
Graph< Airport > readFlights (
            std::string folder )
```

Function to read Flight data from CSV file and add edges to graph of airports Complexity: O(n + v), where n is the number of airports and v is the number of flights in the csv file.

Parameters

folder	The folder where the CSV file is located.

Returns

Graph<Airport> The graph of airports with flights.

Definition at line 72 of file read.cpp.

```
00073 {
00074   // Read airports
00075   Graph<Airport> airports = readAirports(folder);
00076
00077   // Open csv file
00078   std::ifstream file("../dataset/" + folder + "/flights.csv", ios::in);
00079   if (!file.is_open())
00080   {
00081     std::cout « "Error opening flights file" « std::endl;
00082     return airports;
00083   }
00084
00085   // Read and ignore header line
00086   std::string line;
00087   if (!std::getline(file, line))
00088   {
00089     std::cout « "Error reading flights header line" « std::endl;
00090     file.close();
00091     return airports;
00092   }
00093
00094   // Read each line and add edges to graph
00095   while (std::getline(file, line))
00096   {
00097     std::istringstream iss(line);
00098     std::string source, target, airline;
00099
00100     std::getline(iss, source, ',');
00101     std::getline(iss, target, ',');
00102     std::getline(iss, airline, ',');
00103
00104     Airport src = Airport(source);
00105     Airport tgt = Airport(target);
00106
00107     airports.addEdge(src, tgt, 0, airline);
00108   }
00109
00110   // Close file and return the updated graph of airports with flights
00111   file.close();
00112   return airports;
00113 }
```

4.17.2 Variable Documentation

4.17.2.1 airportsHash

```
unordered_map<string, Airport> airportsHash  [extern]
```

Definition at line 3 of file read.cpp.

4.17.2.2 citiesHash

```
unordered_set<string> citiesHash  [extern]
```

Definition at line 4 of file read.cpp.

4.17.2.3 countriesHash

```
unordered_set<string> countriesHash  [extern]
```

Definition at line 5 of file read.cpp.

4.18 read.h

Go to the documentation of this file.

```
00001 // read.h
00002
00003 #ifndef READ_H
00004 #define READ_H
00005
00006 #include "../classes/Airline.h"
00007 #include "../classes/Airport.h"
00008 #include "../classes/Graph.h"
00009 #include <filesystem>
00010 #include <fstream>
00011 #include <iostream>
00012 #include <sstream>
00013 #include <unordered_map>
00014 #include <unordered_set>
00015
00016 extern unordered_map<string, Airport> airportsHash;
00017 extern unordered_set<string> citiesHash;
00018 extern unordered_set<string> countriesHash;
00019
00020 unordered_map<string, Airline> readAirlines(std::string folder);
00021 Graph<Airport> readAirports(std::string folder);
00022 Graph<Airport> readFlights(std::string folder);
00023
00024 #endif // READ_H
```

4.19 src/main.cpp File Reference

```
#include "Menu.h"
```

Functions

- int main (int argc, char const *argv[])

  Main function to run the program and load the database of airports and flights Complexity: O(1)

4.19.1 Function Documentation

4.19.1.1 main()

```
int main (
            int argc,
            char const * argv[] )
```

Main function to run the program and load the database of airports and flights Complexity: O(1)

Parameters

argc	The number of arguments. If no argument is passed, the default folder is "fake".
argv	The arguments (folder name).

Returns

int 0 if the program runs successfully.

Definition at line 10 of file main.cpp.

```
00011 {
00012   if (argc < 2)
00013   {
00014     argv[1] = "original";
00015   }
00016   std::cout « "Loading database..." « std::endl;
00017   Menu(argv[1]);
00018   return 0;
00019 }
```

4.20 main.cpp

Go to the documentation of this file.

```
00001 #include "Menu.h"
00002
00010 int main(int argc, char const *argv[])
00011 {
00012   if (argc < 2)
00013   {
00014     argv[1] = "original";
00015   }
00016   std::cout « "Loading database..." « std::endl;
00017   Menu(argv[1]);
00018   return 0;
00019 }
```

4.21 src/Menu.cpp File Reference

```
#include "Menu.h"
```

Functions

- void Menu (std::string folder)

  Menu function to display the main menu.
- void menuQuantity ()

  Function to handle quantity-related menu options.
- void menuListing ()

  Function to handle listing-related menu options.
- int selectType (std::string arg)

  Function to handle best flights menu options.
- std::string typeAirport (std::string type, int flag)

  Function to input airport code based on the type and flag.
- pair< std::string, std::string > typeCity (std::string type, int flag)

  Function to input city details based on the type and flag.
- pair< double, double > typeCoordinates (std::string type, int flag)

  Function to input coordinates based on the type and flag.
- vector< string > filterAirplanes ()

  Function to filter airplanes on the user input.
- void bestFlights ()

  Function to find best flights based on the user input.
- void menuAirports ()

  Function Menu Airports.
- void menuCountries ()

  Function Menu Countries.
- void menuCities ()

  Function Menu Cities.
- void menuAirlines ()

  Function Menu Airlines.
- void menuFlights ()

  Function Menu Flights.
- void menuDestination ()

  Function Menu Destination.

Variables

- Graph< Airport > airports
- unordered_map< string, Airline > airlines

4.21.1 Function Documentation

4.21.1.1 bestFlights()

```
void bestFlights ( )
```

Function to find best flights based on the user input.

Returns

void

Definition at line 383 of file Menu.cpp.

```
00384 {
00385   vector<string> airplanes = filterAirplanes();
00386
00387   std::string airportOrig;
00388   std::string airportDest;
00389
00390   std::pair<std::string, std::string> cityOrig;
00391   std::pair<std::string, std::string> cityDest;
00392
00393   std::pair<double, double> cordOrig;
00394   std::pair<double, double> cordDest;
00395
00396   int maxDist;
00397   int flagOrigin = selectType("origin");
00398
00399   switch (flagOrigin)
00400   {
00401   case (1):
00402     airportOrig = typeAirport("origin", flagOrigin);
00403     break;
00404   case (2):
00405     cityOrig = typeCity("origin", flagOrigin);
00406     break;
00407   case (3):
00408     cordOrig = typeCoordinates("origin", flagOrigin);
00409     std::cout « "Type Max Distance in (km): " « std::endl;
00410     std::cin » maxDist;
00411     while (std::cin.fail())
00412     {
00413       std::cin.clear();
00414       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00415       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00416       Menu("");
00417     }
00418     break;
00419   case (0):
00420     bestFlights();
00421     break;
00422   default:
00423     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00424     bestFlights();
00425     break;
00426   }
00427
00428   int flagDest = selectType("destination");
00429   switch (flagDest)
00430   {
00431   case (1):
00432     airportDest = typeAirport("destination", flagDest);
00433     break;
00434   case (2):
00435     cityDest = typeCity("destination", flagDest);
00436     break;
00437   case (3):
00438     cordDest = typeCoordinates("destination", flagDest);
00439     std::cout « "Type Max Distance in (km): " « std::endl;
00440     std::cin » maxDist;
00441     while (std::cin.fail())
00442     {
00443       std::cin.clear();
00444       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00445       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00446       Menu("");
00447     }
00448     break;
00449   case (0):
00450     bestFlights();
00451     break;
00452   default:
00453     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00454     bestFlights();
00455     break;
00456   }
00457
00458   switch (flagOrigin)
00459   {
00460   case (1):
00461     switch (flagDest)
00462     {
00463       {
```

```
00464       case (1): // airport to airport
00465         findBestFlights(airports, airportOrig, airportDest, airplanes);
00466         break;
00467       case (2): // airport to city
00468         findBestFlights(airports, cityDest.second, cityDest.first, airportOrig,
00469                         1, airplanes);
00470         break;
00471       case (3): // airport to coordinates
00472         findBestFlights(airports, airportOrig, cordDest.first, cordDest.second,
00473                         maxDist, 1, airplanes);
00474         break;
00475       }
00476     }
00477     break;
00478   case (2):
00479     switch (flagDest)
00480     {
00481       {
00482       case (1): // city to airport
00483         findBestFlights(airports, cityOrig.second, cityOrig.first, airportDest,
00484                         0, airplanes);
00485         break;
00486       case (2): // city to city
00487         findBestFlights(airports, cityOrig.second, cityOrig.first,
00488                         cityDest.second, cityDest.first, airplanes);
00489         break;
00490       case (3): // city to coordinates
00491         findBestFlights(airports, cityOrig.second, cityOrig.first,
00492                         cordDest.first, cordDest.second, maxDist, 0, airplanes);
00493         break;
00494       }
00495     }
00496     break;
00497   case (3):
00498     switch (flagDest)
00499     {
00500       {
00501       case (1): // coordinates to airport
00502         findBestFlights(airports, airportDest, cordOrig.first, cordOrig.second,
00503                         maxDist, 0, airplanes);
00504         break;
00505       case (2): // coordinates to city
00506         findBestFlights(airports, cityDest.second, cityDest.first,
00507                         cordOrig.first, cordOrig.second, maxDist, 1, airplanes);
00508         break;
00509       case (3): // coordinates to coordinates
00510         findBestFlights(airports, cordOrig.first, cordOrig.second,
00511                         cordDest.first, cordDest.second, maxDist, airplanes);
00512         break;
00513       }
00514     }
00515     break;
00516   }
00517
00518   std::cout << "-------------------------------------------------" << std::endl;
00519
00520   std::cout << "Press any key to continue..." << std::endl;
00521   std::cin.ignore();
00522   std::cin.get();
00523   bestFlights();
00524 }
```

4.21.1.2 filterAirplanes()

```
vector< string > filterAirplanes ( )
```

Function to filter airplanes on the user input.

Returns

vector<string> The vector of airplanes to filter

Definition at line 322 of file Menu.cpp.

```
00323 {
00324   vector<string> airplanes;
00325   string line;
00326   int flag;
```

```
00327
00328   system("clear");
00329   std::cout « "Bests flights: " « std::endl;
00330   std::cout « "-------------------------------------------------" « std::endl;
00331
00332   std::cout « "1. Filter by airplanes" « std::endl;
00333   std::cout « "2. Without filter" « std::endl;
00334   std::cout « "0. Back to Main Menu" « std::endl;
00335   std::cout « "-------------------------------------------------" « std::endl;
00336
00337   std::cin » flag;
00338   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00339
00340   while (std::cin.fail())
00341   {
00342     std::cout « "Erro na leitura" « std::endl;
00343     std::cin.clear();
00344     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00345     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00346     filterAirplanes();
00347   }
00348
00349   switch (flag)
00350   {
00351   case (1):
00352   {
00353     std::cout « "Type Airplane Name: " « std::endl;
00354     std::getline(std::cin, line);
00355
00356     std::istringstream iss(line);
00357
00358     string airplane;
00359
00360     while (iss » airplane)
00361     {
00362       airplanes.push_back(airplane);
00363     }
00364     break;
00365   }
00366   case (2):
00367     break;
00368   case (0):
00369     Menu("");
00370     break;
00371   default:
00372     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00373     filterAirplanes();
00374     break;
00375   }
00376   return airplanes;
00377 }
```

4.21.1.3 Menu()

```
void Menu (
            std::string folder )
```

Menu function to display the main menu.

Parameters

folder	The folder name to read the data from.

Returns

void

Definition at line 13 of file Menu.cpp.

```
00014 {
00015   int flag;
00016
00017   if (!folder.empty())
00018   {
```

```
00019     airports = readFlights(folder);
00020     airlines = readAirlines(folder);
00021   }
00022
00023   system("clear");
00024   std::cout « "Welcome to Travel Management:" « std::endl;
00025   std::cout « "-----------------------------------------------" « std::endl;
00026   std::cout « "1. Quantity calculation" « std::endl;
00027   std::cout « "2. Listing" « std::endl;
00028   std::cout « "3. Bests flights" « std::endl;
00029   std::cout « "0. Exit" « std::endl;
00030   std::cout « "-----------------------------------------------" « std::endl;
00031
00032   std::cin » flag;
00033   while (std::cin.fail())
00034   {
00035     std::cin.clear();
00036     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00037     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00038     Menu("");
00039   }
00040
00041   switch (flag)
00042   {
00043   case 1:
00044     menuQuantity();
00045     break;
00046   case 2:
00047     menuListing();
00048     break;
00049   case 3:
00050     bestFlights();
00051     break;
00052   case 0:
00053     exit(0);
00054     break;
00055   default:
00056     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00057     Menu("");
00058     break;
00059   }
00060 }
```

4.21.1.4 menuAirlines()

```
void menuAirlines ( )
```

Function Menu Airlines.

Returns

void

Definition at line 749 of file Menu.cpp.

```
00750 {
00751   int flag;
00752   std::string arg;
00753
00754   system("clear");
00755   std::cout « "Number of Airlines:" « std::endl;
00756   std::cout « "-----------------------------------------------" « std::endl;
00757
00758   std::cout « "1. All Airlines" « std::endl;
00759   std::cout « "2. Airlines by Country" « std::endl;
00760   std::cout « "0. Back to Main Menu" « std::endl;
00761   std::cout « "-----------------------------------------------" « std::endl;
00762
00763   std::cin » flag;
00764   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00765
00766   while (std::cin.fail())
00767   {
00768     std::cin.clear();
00769     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00770     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00771     menuFlights();
00772   }
00773
```

```
00774   switch (flag)
00775   {
00776   case 1:
00777     std::cout « "All Airlines: " « airlines.size()
00778               « std::endl;
00779     std::cout « "-------------------------------------------------"
00780               « std::endl;
00781     std::cout « "Press any key to continue..." « std::endl;
00782     std::cin.ignore();
00783     std::cin.get();
00784     menuQuantity();
00785     break;
00786   case 2:
00787     std::cout « "Type Country Name: " « std::endl;
00788     std::getline(std::cin, arg);
00789
00790     while (countriesHash.find(arg) == countriesHash.end())
00791     {
00792       std::cout « "Country not found" « std::endl;
00793       std::cout « "Type Country Name: " « std::endl;
00794       std::getline(std::cin, arg);
00795     }
00796
00797     std::cout « "Airlines in " « arg « ": " « quantityAirlinesCountry(airlines, arg)
00798               « std::endl;
00799     std::cout « "-------------------------------------------------"
00800               « std::endl;
00801     std::cout « "Press any key to continue..." « std::endl;
00802     std::cin.ignore();
00803     std::cin.get();
00804     menuQuantity();
00805     break;
00806   case 0:
00807     menuQuantity();
00808     break;
00809   default:
00810     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00811     menuAirlines();
00812     break;
00813   }
00814 }
```

4.21.1.5 menuAirports()

```
void menuAirports ( )
```

Function Menu Airports.

Returns

void

Definition at line 530 of file Menu.cpp.

```
00531 {
00532   int flag;
00533   std::string arg;
00534   std::string arg2;
00535
00536   system("clear");
00537   std::cout « "Number of Airports:" « std::endl;
00538   std::cout « "-------------------------------------------------" « std::endl;
00539
00540   std::cout « "1. All Airports" « std::endl;
00541   std::cout « "2. Airports by Country" « std::endl;
00542   std::cout « "3. Airports by City" « std::endl;
00543   std::cout « "0. Back to Main Menu" « std::endl;
00544   std::cout « "-------------------------------------------------" « std::endl;
00545
00546   std::cin » flag;
00547   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00548
00549   while (std::cin.fail())
00550   {
00551     std::cin.clear();
00552     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00553     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00554     menuFlights();
```

```
00555   }
00556
00557   switch (flag)
00558   {
00559   case 1:
00560     std::cout « "All Airports: " « quantityAirports(airports) « std::endl;
00561     std::cout « "-------------------------------------------------"
00562               « std::endl;
00563     std::cout « "Press any key to continue..." « std::endl;
00564     std::cin.ignore();
00565     std::cin.get();
00566     menuQuantity();
00567     break;
00568   case 2:
00569     std::cout « "Type Country Name: " « std::endl;
00570     std::getline(std::cin, arg);
00571
00572     while (countriesHash.find(arg) == countriesHash.end())
00573     {
00574       std::cout « "Country not found" « std::endl;
00575       std::cout « "Type Country Name: " « std::endl;
00576       std::getline(std::cin, arg);
00577     }
00578
00579     std::cout « "Airports in " « arg « ": "
00580               « quantityAirportsCountry(airports, arg) « std::endl;
00581     std::cout « "-------------------------------------------------"
00582               « std::endl;
00583     std::cout « "Press any key to continue..." « std::endl;
00584     std::cin.ignore();
00585     std::cin.get();
00586     menuQuantity();
00587     break;
00588   case 3:
00589     std::cout « "Type City Name: " « std::endl;
00590     std::getline(std::cin, arg);
00591
00592     while (citiesHash.find(arg) == citiesHash.end())
00593     {
00594       std::cout « "City not found" « std::endl;
00595       std::cout « "Type City Name: " « std::endl;
00596       std::getline(std::cin, arg);
00597     }
00598
00599     std::cout « "Type Country Name: " « std::endl;
00600     std::getline(std::cin, arg2);
00601
00602     while (countriesHash.find(arg2) == countriesHash.end())
00603     {
00604       std::cout « "Country not found" « std::endl;
00605       std::cout « "Type Country Name: " « std::endl;
00606       std::getline(std::cin, arg2);
00607     }
00608
00609     std::cout « "Airports in " « arg « ", " « arg2 « ": "
00610               « quantityAirportsCity(airports, arg) « std::endl;
00611     std::cout « "-------------------------------------------------"
00612               « std::endl;
00613     std::cout « "Press any key to continue..." « std::endl;
00614     std::cin.ignore();
00615     std::cin.get();
00616     menuQuantity();
00617     break;
00618   case 0:
00619     menuQuantity();
00620     break;
00621   default:
00622     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00623     menuAirports();
00624     break;
00625   }
00626 }
```

4.21.1.6 menuCities()

```
void menuCities ( )
```

Function Menu Cities.

Returns

void

Definition at line 679 of file Menu.cpp.

```
00680 {
00681   int flag;
00682   std::string arg;
00683
00684   system("clear");
00685   std::cout « "Number of Cities:" « std::endl;
00686   std::cout « "-------------------------------------------------" « std::endl;
00687
00688   std::cout « "1. All Cities" « std::endl;
00689   std::cout « "2. Cities by Country" « std::endl;
00690   std::cout « "0. Back to Main Menu" « std::endl;
00691   std::cout « "-------------------------------------------------" « std::endl;
00692
00693   std::cin » flag;
00694   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00695
00696   while (std::cin.fail())
00697   {
00698     std::cin.clear();
00699     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00700     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00701     menuFlights();
00702   }
00703
00704   switch (flag)
00705   {
00706   case 1:
00707     std::cout « "All Cities: " « citiesHash.size() « std::endl;
00708     std::cout « "-------------------------------------------------"
00709               « std::endl;
00710     std::cout « "Press any key to continue..." « std::endl;
00711     std::cin.ignore();
00712     std::cin.get();
00713     menuQuantity();
00714     break;
00715   case 2:
00716     std::cout « "Type Country Name: " « std::endl;
00717     std::getline(std::cin, arg);
00718
00719     while (countriesHash.find(arg) == countriesHash.end())
00720     {
00721       std::cout « "Country not found" « std::endl;
00722       std::cout « "Type Country Name: " « std::endl;
00723       std::getline(std::cin, arg);
00724     }
00725
00726     std::cout « "Cities in " « arg « ": "
00727               « quantityCitiesCountry(airports, arg) « std::endl;
00728     std::cout « "-------------------------------------------------"
00729               « std::endl;
00730     std::cout « "Press any key to continue..." « std::endl;
00731     std::cin.ignore();
00732     std::cin.get();
00733     menuQuantity();
00734     break;
00735   case 0:
00736     menuQuantity();
00737     break;
00738   default:
00739     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00740     menuCities();
00741     break;
00742   }
00743 }
```

4.21.1.7 menuCountries()

```
void menuCountries ( )
```

Function Menu Countries.

Returns

void

Definition at line 632 of file Menu.cpp.

```
00633 {
00634   int flag;
00635   std::string arg;
00636
00637   system("clear");
00638   std::cout « "Number of Countries:" « std::endl;
00639   std::cout « "-------------------------------------------------" « std::endl;
00640
00641   std::cout « "1. All Countries" « std::endl;
00642   std::cout « "0. Back to Main Menu" « std::endl;
00643   std::cout « "-------------------------------------------------" « std::endl;
00644
00645   std::cin » flag;
00646   while (std::cin.fail())
00647   {
00648     std::cin.clear();
00649     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00650     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00651     menuFlights();
00652   }
00653
00654   switch (flag)
00655   {
00656   case 1:
00657     std::cout « "All Countries: " « countriesHash.size() « std::endl;
00658     std::cout « "-------------------------------------------------"
00659               « std::endl;
00660     std::cout « "Press any key to continue..." « std::endl;
00661     std::cin.ignore();
00662     std::cin.get();
00663     menuQuantity();
00664     break;
00665   case 0:
00666     menuQuantity();
00667     break;
00668   default:
00669     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00670     menuCountries();
00671     break;
00672   }
00673 }
```

4.21.1.8 menuDestination()

```
void menuDestination ( )
```

Function Menu Destination.

Returns

void

Definition at line 956 of file Menu.cpp.

```
00957 {
00958   int flag;
00959   std::string arg;
00960   int stop;
00961
00962   system("clear");
00963   std::cout « "Number of Destinations:" « std::endl;
00964   std::cout « "-------------------------------------------------" « std::endl;
00965
00966   std::cout « "1. Unlimited Stops (by airport)" « std::endl;
00967   std::cout « "2. Limited Stops" « std::endl;
00968   std::cout « "3. Max destinations" « std::endl;
00969   std::cout « "0. Back to Main Menu" « std::endl;
00970   std::cout « "-------------------------------------------------" « std::endl;
00971
00972   std::cin » flag;
00973   while (std::cin.fail())
00974   {
```

```
00975     std::cin.clear();
00976     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00977     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00978     menuDestination();
00979   }
00980
00981   switch (flag)
00982   {
00983   case 1:
00984     std::cout « "Type Airport Code: " « std::endl;
00985     std::cin » arg;
00986
00987     if (airportsHash.find(arg) == airportsHash.end())
00988     {
00989       std::cout « "Airport not found" « std::endl;
00990       std::cout « "-------------------------------------------------"
00991                 « std::endl;
00992       std::cout « "Press any key to continue..." « std::endl;
00993       std::cin.ignore();
00994       std::cin.get();
00995       menuDestination();
00996     }
00997
00998     std::cout « "Destination in " « arg « ": "
00999               « quantityDestinationsAirport(airports, arg) « std::endl;
01000     std::cout « "-------------------------------------------------"
01001               « std::endl;
01002     std::cout « "Press any key to continue..." « std::endl;
01003     std::cin.ignore();
01004     std::cin.get();
01005     menuQuantity();
01006     break;
01007   case 2:
01008     std::cout « "Type Airport Code: " « std::endl;
01009     std::cin » arg;
01010
01011     if (airportsHash.find(arg) == airportsHash.end())
01012     {
01013       std::cout « "Airport not found" « std::endl;
01014       std::cout « "-------------------------------------------------"
01015                 « std::endl;
01016       std::cout « "Press any key to continue..." « std::endl;
01017       std::cin.ignore();
01018       std::cin.get();
01019       menuDestination();
01020     }
01021
01022     std::cout « "Type Stops Number: " « std::endl;
01023     std::cin » stop;
01024     while (std::cin.fail())
01025     {
01026       std::cin.clear();
01027       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
01028       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
01029       Menu("");
01030     }
01031     std::cout « "Number of Countries: "
01032               « quantityDestinationLimitedStop(airports, arg, stop)
01033               « std::endl;
01034     std::cout « "-------------------------------------------------"
01035               « std::endl;
01036     std::cout « "Press any key to continue..." « std::endl;
01037     std::cin.ignore();
01038     std::cin.get();
01039     menuDestination();
01040   case 3:
01041     std::cout « "Max number of flights: " « quantityDestinationMax(airports)
01042               « " stops" « std::endl;
01043     std::cout « "-------------------------------------------------"
01044               « std::endl;
01045     std::cout « "Press any key to continue..." « std::endl;
01046     std::cin.ignore();
01047     std::cin.get();
01048     menuDestination();
01049   case 0:
01050     Menu("");
01051     break;
01052   default:
01053     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
01054     menuDestination();
01055     break;
01056   }
01057 }
```

4.21.1.9 menuFlights()

```
void menuFlights ( )
```

Function Menu Flights.

Returns

void

Definition at line 820 of file Menu.cpp.

```
00821 {
00822   int flag;
00823   std::string arg;
00824   std::string arg2;
00825 
00826   system("clear");
00827   std::cout « "Number of flights:" « std::endl;
00828   std::cout « "-------------------------------------------------" « std::endl;
00829 
00830   std::cout « "1. All flights" « std::endl;
00831   std::cout « "2. Flights by Origin Airport" « std::endl;
00832   std::cout « "3. Flights by Origin Country" « std::endl;
00833   std::cout « "4. Flights by City" « std::endl;
00834   std::cout « "5. Flights by Airline" « std::endl;
00835   std::cout « "0. Back to Main Menu" « std::endl;
00836   std::cout « "-------------------------------------------------" « std::endl;
00837 
00838   std::cin » flag;
00839   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00840 
00841   while (std::cin.fail())
00842   {
00843     std::cin.clear();
00844     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00845     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00846     menuFlights();
00847   }
00848 
00849   switch (flag)
00850   {
00851   case 1:
00852     std::cout « "All flights: " « quantityFlights(airports) « std::endl;
00853     std::cout « "-------------------------------------------------"
00854               « std::endl;
00855     std::cout « "Press any key to continue..." « std::endl;
00856     std::cin.ignore();
00857     std::cin.get();
00858     menuQuantity();
00859     break;
00860   case 2:
00861     std::cout « "Type Origin Airport Code: " « std::endl;
00862     std::cin » arg;
00863 
00864     while (airportsHash.find(arg) == airportsHash.end())
00865     {
00866       std::cout « "Airport not found" « std::endl;
00867       std::cout « "Type Origin Airport Code: " « std::endl;
00868       std::cin » arg;
00869     }
00870 
00871     std::cout « "Flights in " « arg « ": "
00872               « quantityFlightsAirport(airports, arg) « std::endl;
00873     std::cout « "-------------------------------------------------"
00874               « std::endl;
00875     std::cout « "Press any key to continue..." « std::endl;
00876     std::cin.ignore();
00877     std::cin.get();
00878     menuQuantity();
00879     break;
00880   case 3:
00881     std::cout « "Type Origin Country Code: " « std::endl;
00882     std::getline(std::cin, arg);
00883 
00884     while (countriesHash.find(arg) == countriesHash.end())
00885     {
00886       std::cout « "Country not found" « std::endl;
00887       std::cout « "Type Origin Country Code: " « std::endl;
00888       std::getline(std::cin, arg);
00889     }
```

```
00890
00891     std::cout « "Flights in " « arg « ": "
00892               « quantityFlightsCountry(airports, arg) « std::endl;
00893     std::cout « "-------------------------------------------------"
00894               « std::endl;
00895     std::cout « "Press any key to continue..." « std::endl;
00896     std::cin.ignore();
00897     std::cin.get();
00898     menuQuantity();
00899     break;
00900   case 4:
00901     std::cout « "Type City Name: " « std::endl;
00902     std::getline(std::cin, arg);
00903
00904     while (citiesHash.find(arg) == citiesHash.end())
00905     {
00906       std::cout « "City not found" « std::endl;
00907       std::cout « "Type City Name: " « std::endl;
00908       std::getline(std::cin, arg);
00909     }
00910
00911     std::cout « "Type Country Name: " « std::endl;
00912     std::getline(std::cin, arg2);
00913
00914     while (countriesHash.find(arg2) == countriesHash.end())
00915     {
00916       std::cout « "Country not found" « std::endl;
00917       std::cout « "Type Country Name: " « std::endl;
00918       std::getline(std::cin, arg2);
00919     }
00920
00921     std::cout « "Flights in " « arg « ": "
00922               « quantityFlightsCity(airports, arg) « std::endl;
00923     std::cout « "-------------------------------------------------"
00924               « std::endl;
00925     std::cout « "Press any key to continue..." « std::endl;
00926     std::cin.ignore();
00927     std::cin.get();
00928     menuQuantity();
00929     break;
00930   case 5:
00931     std::cout « "Type Airline Code: " « std::endl;
00932     std::cin » arg;
00933     std::cout « "Flights in " « arg « ": "
00934               « quantityFlightsAirline(airports, arg) « std::endl;
00935     std::cout « "-------------------------------------------------"
00936               « std::endl;
00937     std::cout « "Press any key to continue..." « std::endl;
00938     std::cin.ignore();
00939     std::cin.get();
00940     menuQuantity();
00941     break;
00942   case 0:
00943     menuQuantity();
00944     break;
00945   default:
00946     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00947     menuFlights();
00948     break;
00949   }
00950 }
```

4.21.1.10 menuListing()

```
void menuListing ( )
```

Function to handle listing-related menu options.

Returns

void

Definition at line 132 of file Menu.cpp.

```
00133 {
00134   int flag;
00135   int arg;
00136
```

```
00137   system("clear");
00138   std::cout « "Listing Menu: " « std::endl;
00139   std::cout « "-------------------------------------------------" « std::endl;
00140
00141   std::cout « "1. Ranking Airports (more landings and takeoffs)" « std::endl;
00142   std::cout « "2. Connecting airports" « std::endl;
00143   std::cout « "0. Back to Main Menu" « std::endl;
00144   std::cout « "-------------------------------------------------" « std::endl;
00145
00146   std::cin » flag;
00147   while (std::cin.fail())
00148   {
00149     std::cin.clear();
00150     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00151     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00152     menuListing();
00153   }
00154
00155   switch (flag)
00156   {
00157   case (1):
00158     system("clear");
00159     std::cout « "Type Airports Number: " « std::endl;
00160     std::cin » arg;
00161     std::cout « "-------------------------------------------------"
00162               « std::endl;
00163     std::cout « "Ranking Airports: " « std::endl;
00164     std::cout « "-------------------------------------------------"
00165               « std::endl;
00166     rankingAirports(airports, arg);
00167     std::cout « "-------------------------------------------------"
00168               « std::endl;
00169     std::cout « "Press any key to continue..." « std::endl;
00170     std::cin.ignore();
00171     std::cin.get();
00172     menuListing();
00173     break;
00174   case (2):
00175     system("clear");
00176     std::cout « "Connecting airports: " « std::endl;
00177     std::cout « "-------------------------------------------------"
00178               « std::endl;
00179     // getArticulations(airports);
00180     findArticulationPoints(airports);
00181     // connectedComponents(airports);
00182     std::cout « "-------------------------------------------------"
00183               « std::endl;
00184     std::cout « "Press any key to continue..." « std::endl;
00185     std::cin.ignore();
00186     std::cin.get();
00187     menuListing();
00188     break;
00189   case (0):
00190     Menu("");
00191     break;
00192   default:
00193     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00194     menuListing();
00195     break;
00196   }
00197 }
```

4.21.1.11 menuQuantity()

```
void menuQuantity ( )
```

Function to handle quantity-related menu options.

Returns

void

Definition at line 66 of file Menu.cpp.

```
00067 {
00068   int flag;
00069
00070   system("clear");
```

```
00071   std::cout « "Quantity Calculation Menu" « std::endl;
00072   std::cout « "-------------------------------------------------" « std::endl;
00073
00074   std::cout « "1. Number of airports" « std::endl;
00075   std::cout « "2. Number of countries" « std::endl;
00076   std::cout « "3. Number of cities" « std::endl;
00077   std::cout « "4. Number of airlines" « std::endl;
00078   std::cout « "5. Number of flights" « std::endl;
00079   std::cout « "6. Number of destinations" « std::endl;
00080   std::cout « "0. Back to Main Menu" « std::endl;
00081   std::cout « "-------------------------------------------------" « std::endl;
00082
00083   std::cin » flag;
00084   while (std::cin.fail())
00085   {
00086     std::cin.clear();
00087     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00088     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00089     menuQuantity();
00090   }
00091
00092   switch (flag)
00093   {
00094   case 1:
00095     system("clear");
00096     menuAirports();
00097     break;
00098   case 2:
00099     system("clear");
00100     menuCountries();
00101     break;
00102   case 3:
00103     system("clear");
00104     menuCities();
00105     break;
00106   case 4:
00107     system("clear");
00108     menuAirlines();
00109     break;
00110   case 5:
00111     system("clear");
00112     menuFlights();
00113     break;
00114   case 6:
00115     system("clear");
00116     menuDestination();
00117     break;
00118   case 0:
00119     Menu("");
00120     break;
00121   default:
00122     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00123     menuQuantity();
00124     break;
00125   }
00126 }
```

4.21.1.12 selectType()

```
int selectType (
            std::string arg )
```

Function to handle best flights menu options.

Parameters

arg	The type of the airport (origin or destination)

Returns

void

Definition at line 204 of file Menu.cpp.

```
00205 {
00206   int flag;
00207
00208   system("clear");
00209   std::cout « "Bests Flights: " « std::endl;
00210   std::cout « "Select your " « arg « " option:" « std::endl;
00211   std::cout « "-----------------------------------------------" « std::endl;
00212
00213   std::cout « "1. By airports" « std::endl;
00214   std::cout « "2. By cities" « std::endl;
00215   std::cout « "3. By coordinates" « std::endl;
00216   std::cout « "0. Back to Best Flights" « std::endl;
00217   std::cout « "-----------------------------------------------" « std::endl;
00218
00219   std::cin » flag;
00220   while (std::cin.fail())
00221   {
00222     std::cin.clear();
00223     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00224     selectType(arg);
00225   }
00226   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00227
00228   return flag;
00229 }
```

4.21.1.13 typeAirport()

```
std::string typeAirport (
              std::string type,
              int flag )
```

Function to input airport code based on the type and flag.

Parameters

type	The type of the airport (origin or destination)
flag	The flag to select the type of filter (airport, city or coordinates)

Returns

string The airport code

Definition at line 237 of file Menu.cpp.

```
00238 {
00239   std::string arg;
00240   switch (flag)
00241   {
00242   case (1):
00243     std::cout « "Type Code Airport of " « type « ": " « std::endl;
00244     std::cin » arg;
00245     while (airportsHash.find(arg) == airportsHash.end())
00246     {
00247       std::cout « "Airport not found" « std::endl;
00248       std::cout « "Type Code Airport of " « type « ": " « std::endl;
00249       std::cin » arg;
00250     }
00251     break;
00252   }
00253   return arg;
00254 }
```

4.21.1.14 typeCity()

```
pair< std::string, std::string > typeCity (
              std::string type,
              int flag )
```

Function to input city details based on the type and flag.

Parameters

type	The type of the airport (origin or destination)
flag	The flag to select the type of filter (airport, city or coordinates)

Returns

pair<string, string> The city name and country name

Definition at line 262 of file Menu.cpp.

```
00263 {
00264   std::string arg1;
00265   std::string arg2;
00266   switch (flag)
00267   {
00268   case (2):
00269     std::cout « "Type City Name " « type « ": " « std::endl;
00270     std::getline(std::cin, arg1);
00271
00272     while (citiesHash.find(arg1) == citiesHash.end())
00273     {
00274       std::cout « "City not found" « std::endl;
00275       std::cout « "Type City Name " « type « ": " « std::endl;
00276       std::getline(std::cin, arg1);
00277     }
00278
00279     std::cout « "Type Country Name " « type « ": " « std::endl;
00280     std::getline(std::cin, arg2);
00281
00282     while (countriesHash.find(arg2) == countriesHash.end())
00283     {
00284       std::cout « "Country not found" « std::endl;
00285       std::cout « "Type Country Name " « type « ": " « std::endl;
00286       std::getline(std::cin, arg2);
00287     }
00288
00289     break;
00290   }
00291   return make_pair(arg1, arg2);
00292 }
```

4.21.1.15 typeCoordinates()

```
pair< double, double > typeCoordinates (
            std::string type,
            int flag )
```

Function to input coordinates based on the type and flag.

Parameters

type	The type of the airport (origin or destination)
flag	The flag to select the type of filter (airport, city or coordinates)

Returns

pair<double, double> The latitude and longitude

Definition at line 300 of file Menu.cpp.

```
00301 {
00302   double arg1;
00303   double arg2;
00304   switch (flag)
```

```
00305   {
00306   case (3):
00307     std::cout « "Type Latitude of " « type « ": " « std::endl;
00308     std::cin » arg1;
00309     std::cout « "Type Longitude of " « type « ": " « std::endl;
00310     std::cin » arg2;
00311     break;
00312   }
00313   return make_pair(arg1, arg2);
00314 }
```

4.21.2 Variable Documentation

4.21.2.1 airlines

```
unordered_map<string, Airline> airlines
```

Definition at line 5 of file Menu.cpp.

4.21.2.2 airports

```
Graph<Airport> airports
```

Definition at line 4 of file Menu.cpp.

4.22 Menu.cpp

Go to the documentation of this file.

```
00001 #include "Menu.h"
00002
00003 // Global graph to store airport data
00004 Graph<Airport> airports;
00005 unordered_map<string, Airline> airlines;
00006
00012 // Complexity: O(1)
00013 void Menu(std::string folder)
00014 {
00015   int flag;
00016
00017   if (!folder.empty())
00018   {
00019     airports = readFlights(folder);
00020     airlines = readAirlines(folder);
00021   }
00022
00023   system("clear");
00024   std::cout « "Welcome to Travel Management:" « std::endl;
00025   std::cout « "-----------------------------------------------" « std::endl;
00026   std::cout « "1. Quantity calculation" « std::endl;
00027   std::cout « "2. Listing" « std::endl;
00028   std::cout « "3. Bests flights" « std::endl;
00029   std::cout « "0. Exit" « std::endl;
00030   std::cout « "-----------------------------------------------" « std::endl;
00031
00032   std::cin » flag;
00033   while (std::cin.fail())
00034   {
00035     std::cin.clear();
00036     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00037     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00038     Menu("");
00039   }
00040
00041   switch (flag)
00042   {
00043   case 1:
00044     menuQuantity();
00045     break;
```

```
00046   case 2:
00047     menuListing();
00048     break;
00049   case 3:
00050     bestFlights();
00051     break;
00052   case 0:
00053     exit(0);
00054     break;
00055   default:
00056     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00057     Menu("");
00058     break;
00059   }
00060 }
00061
00066 void menuQuantity()
00067 {
00068   int flag;
00069
00070   system("clear");
00071   std::cout « "Quantity Calculation Menu" « std::endl;
00072   std::cout « "-----------------------------------------------" « std::endl;
00073
00074   std::cout « "1. Number of airports" « std::endl;
00075   std::cout « "2. Number of countries" « std::endl;
00076   std::cout « "3. Number of cities" « std::endl;
00077   std::cout « "4. Number of airlines" « std::endl;
00078   std::cout « "5. Number of flights" « std::endl;
00079   std::cout « "6. Number of destinations" « std::endl;
00080   std::cout « "0. Back to Main Menu" « std::endl;
00081   std::cout « "-----------------------------------------------" « std::endl;
00082
00083   std::cin » flag;
00084   while (std::cin.fail())
00085   {
00086     std::cin.clear();
00087     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00088     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00089     menuQuantity();
00090   }
00091
00092   switch (flag)
00093   {
00094   case 1:
00095     system("clear");
00096     menuAirports();
00097     break;
00098   case 2:
00099     system("clear");
00100     menuCountries();
00101     break;
00102   case 3:
00103     system("clear");
00104     menuCities();
00105     break;
00106   case 4:
00107     system("clear");
00108     menuAirlines();
00109     break;
00110   case 5:
00111     system("clear");
00112     menuFlights();
00113     break;
00114   case 6:
00115     system("clear");
00116     menuDestination();
00117     break;
00118   case 0:
00119     Menu("");
00120     break;
00121   default:
00122     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00123     menuQuantity();
00124     break;
00125   }
00126 }
00127
00132 void menuListing()
00133 {
00134   int flag;
00135   int arg;
00136
00137   system("clear");
00138   std::cout « "Listing Menu: " « std::endl;
00139   std::cout « "-----------------------------------------------" « std::endl;
00140
```

```
00141   std::cout « "1. Ranking Airports (more landings and takeoffs)" « std::endl;
00142   std::cout « "2. Connecting airports" « std::endl;
00143   std::cout « "0. Back to Main Menu" « std::endl;
00144   std::cout « "------------------------------------------------" « std::endl;
00145
00146   std::cin » flag;
00147   while (std::cin.fail())
00148   {
00149     std::cin.clear();
00150     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00151     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00152     menuListing();
00153   }
00154
00155   switch (flag)
00156   {
00157   case (1):
00158     system("clear");
00159     std::cout « "Type Airports Number: " « std::endl;
00160     std::cin » arg;
00161     std::cout « "------------------------------------------------"
00162               « std::endl;
00163     std::cout « "Ranking Airports: " « std::endl;
00164     std::cout « "------------------------------------------------"
00165               « std::endl;
00166     rankingAirports(airports, arg);
00167     std::cout « "------------------------------------------------"
00168               « std::endl;
00169     std::cout « "Press any key to continue..." « std::endl;
00170     std::cin.ignore();
00171     std::cin.get();
00172     menuListing();
00173     break;
00174   case (2):
00175     system("clear");
00176     std::cout « "Connecting airports: " « std::endl;
00177     std::cout « "------------------------------------------------"
00178               « std::endl;
00179     // getArticulations(airports);
00180     findArticulationPoints(airports);
00181     // connectedComponents(airports);
00182     std::cout « "------------------------------------------------"
00183               « std::endl;
00184     std::cout « "Press any key to continue..." « std::endl;
00185     std::cin.ignore();
00186     std::cin.get();
00187     menuListing();
00188     break;
00189   case (0):
00190     Menu("");
00191     break;
00192   default:
00193     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00194     menuListing();
00195     break;
00196   }
00197 }
00198
00204 int selectType(std::string arg)
00205 {
00206   int flag;
00207
00208   system("clear");
00209   std::cout « "Bests Flights: " « std::endl;
00210   std::cout « "Select your " « arg « " option:" « std::endl;
00211   std::cout « "------------------------------------------------" « std::endl;
00212
00213   std::cout « "1. By airports" « std::endl;
00214   std::cout « "2. By cities" « std::endl;
00215   std::cout « "3. By coordinates" « std::endl;
00216   std::cout « "0. Back to Best Flights" « std::endl;
00217   std::cout « "------------------------------------------------" « std::endl;
00218
00219   std::cin » flag;
00220   while (std::cin.fail())
00221   {
00222     std::cin.clear();
00223     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00224     selectType(arg);
00225   }
00226   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00227
00228   return flag;
00229 }
00230
00237 std::string typeAirport(std::string type, int flag)
00238 {
```

```
00239   std::string arg;
00240   switch (flag)
00241   {
00242   case (1):
00243     std::cout « "Type Code Airport of " « type « ": " « std::endl;
00244     std::cin » arg;
00245     while (airportsHash.find(arg) == airportsHash.end())
00246     {
00247       std::cout « "Airport not found" « std::endl;
00248       std::cout « "Type Code Airport of " « type « ": " « std::endl;
00249       std::cin » arg;
00250     }
00251     break;
00252   }
00253   return arg;
00254 }
00255
00262 pair<std::string, std::string> typeCity(std::string type, int flag)
00263 {
00264   std::string arg1;
00265   std::string arg2;
00266   switch (flag)
00267   {
00268   case (2):
00269     std::cout « "Type City Name " « type « ": " « std::endl;
00270     std::getline(std::cin, arg1);
00271
00272     while (citiesHash.find(arg1) == citiesHash.end())
00273     {
00274       std::cout « "City not found" « std::endl;
00275       std::cout « "Type City Name " « type « ": " « std::endl;
00276       std::getline(std::cin, arg1);
00277     }
00278
00279     std::cout « "Type Country Name " « type « ": " « std::endl;
00280     std::getline(std::cin, arg2);
00281
00282     while (countriesHash.find(arg2) == countriesHash.end())
00283     {
00284       std::cout « "Country not found" « std::endl;
00285       std::cout « "Type Country Name " « type « ": " « std::endl;
00286       std::getline(std::cin, arg2);
00287     }
00288
00289     break;
00290   }
00291   return make_pair(arg1, arg2);
00292 }
00293
00300 pair<double, double> typeCoordinates(std::string type, int flag)
00301 {
00302   double arg1;
00303   double arg2;
00304   switch (flag)
00305   {
00306   case (3):
00307     std::cout « "Type Latitude of " « type « ": " « std::endl;
00308     std::cin » arg1;
00309     std::cout « "Type Longitude of " « type « ": " « std::endl;
00310     std::cin » arg2;
00311     break;
00312   }
00313   return make_pair(arg1, arg2);
00314 }
00315
00316 // Function to filter airplanes on the user input
00317
00322 vector<string> filterAirplanes()
00323 {
00324   vector<string> airplanes;
00325   string line;
00326   int flag;
00327
00328   system("clear");
00329   std::cout « "Bests flights: " « std::endl;
00330   std::cout « "-------------------------------------------------" « std::endl;
00331
00332   std::cout « "1. Filter by airplanes" « std::endl;
00333   std::cout « "2. Without filter" « std::endl;
00334   std::cout « "0. Back to Main Menu" « std::endl;
00335   std::cout « "-------------------------------------------------" « std::endl;
00336
00337   std::cin » flag;
00338   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00339
00340   while (std::cin.fail())
00341   {
```

```
00342     std::cout « "Erro na leitura" « std::endl;
00343     std::cin.clear();
00344     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00345     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00346     filterAirplanes();
00347   }
00348
00349   switch (flag)
00350   {
00351   case (1):
00352   {
00353     std::cout « "Type Airplane Name: " « std::endl;
00354     std::getline(std::cin, line);
00355
00356     std::istringstream iss(line);
00357
00358     string airplane;
00359
00360     while (iss » airplane)
00361     {
00362       airplanes.push_back(airplane);
00363     }
00364     break;
00365   }
00366   case (2):
00367     break;
00368   case (0):
00369     Menu("");
00370     break;
00371   default:
00372     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00373     filterAirplanes();
00374     break;
00375   }
00376   return airplanes;
00377 }
00378
00383 void bestFlights()
00384 {
00385   vector<string> airplanes = filterAirplanes();
00386
00387   std::string airportOrig;
00388   std::string airportDest;
00389
00390   std::pair<std::string, std::string> cityOrig;
00391   std::pair<std::string, std::string> cityDest;
00392
00393   std::pair<double, double> cordOrig;
00394   std::pair<double, double> cordDest;
00395
00396   int maxDist;
00397   int flagOrigin = selectType("origin");
00398
00399   switch (flagOrigin)
00400   {
00401   case (1):
00402     airportOrig = typeAirport("origin", flagOrigin);
00403     break;
00404   case (2):
00405     cityOrig = typeCity("origin", flagOrigin);
00406     break;
00407   case (3):
00408     cordOrig = typeCoordinates("origin", flagOrigin);
00409     std::cout « "Type Max Distance in (km): " « std::endl;
00410     std::cin » maxDist;
00411     while (std::cin.fail())
00412     {
00413       std::cin.clear();
00414       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00415       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00416       Menu("");
00417     }
00418     break;
00419   case (0):
00420     bestFlights();
00421     break;
00422   default:
00423     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00424     bestFlights();
00425     break;
00426   }
00427
00428   int flagDest = selectType("destination");
00429   switch (flagDest)
00430   {
00431   case (1):
00432     airportDest = typeAirport("destination", flagDest);
```

```
00433     break;
00434   case (2):
00435     cityDest = typeCity("destination", flagDest);
00436     break;
00437   case (3):
00438     cordDest = typeCoordinates("destination", flagDest);
00439     std::cout « "Type Max Distance in (km): " « std::endl;
00440     std::cin » maxDist;
00441     while (std::cin.fail())
00442     {
00443       std::cin.clear();
00444       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00445       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00446       Menu("");
00447     }
00448     break;
00449   case (0):
00450     bestFlights();
00451     break;
00452   default:
00453     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00454     bestFlights();
00455     break;
00456   }
00457
00458   switch (flagOrigin)
00459   {
00460   case (1):
00461     switch (flagDest)
00462     {
00463       {
00464       case (1): // airport to airport
00465         findBestFlights(airports, airportOrig, airportDest, airplanes);
00466         break;
00467       case (2): // airport to city
00468         findBestFlights(airports, cityDest.second, cityDest.first, airportOrig,
00469                         1, airplanes);
00470         break;
00471       case (3): // airport to coordinates
00472         findBestFlights(airports, airportOrig, cordDest.first, cordDest.second,
00473                         maxDist, 1, airplanes);
00474         break;
00475       }
00476     }
00477     break;
00478   case (2):
00479     switch (flagDest)
00480     {
00481       {
00482       case (1): // city to airport
00483         findBestFlights(airports, cityOrig.second, cityOrig.first, airportDest,
00484                         0, airplanes);
00485         break;
00486       case (2): // city to city
00487         findBestFlights(airports, cityOrig.second, cityOrig.first,
00488                         cityDest.second, cityDest.first, airplanes);
00489         break;
00490       case (3): // city to coordinates
00491         findBestFlights(airports, cityOrig.second, cityOrig.first,
00492                         cordDest.first, cordDest.second, maxDist, 0, airplanes);
00493         break;
00494       }
00495     }
00496     break;
00497   case (3):
00498     switch (flagDest)
00499     {
00500       {
00501       case (1): // coordinates to airport
00502         findBestFlights(airports, airportDest, cordOrig.first, cordOrig.second,
00503                         maxDist, 0, airplanes);
00504         break;
00505       case (2): // coordinates to city
00506         findBestFlights(airports, cityDest.second, cityDest.first,
00507                         cordOrig.first, cordOrig.second, maxDist, 1, airplanes);
00508         break;
00509       case (3): // coordinates to coordinates
00510         findBestFlights(airports, cordOrig.first, cordOrig.second,
00511                         cordDest.first, cordDest.second, maxDist, airplanes);
00512         break;
00513       }
00514     }
00515     break;
00516   }
00517
00518   std::cout « "-----------------------------------------------" « std::endl;
00519
```

```
00520   std::cout « "Press any key to continue..." « std::endl;
00521   std::cin.ignore();
00522   std::cin.get();
00523   bestFlights();
00524 }
00525
00530 void menuAirports()
00531 {
00532   int flag;
00533   std::string arg;
00534   std::string arg2;
00535
00536   system("clear");
00537   std::cout « "Number of Airports:" « std::endl;
00538   std::cout « "-------------------------------------------------" « std::endl;
00539
00540   std::cout « "1. All Airports" « std::endl;
00541   std::cout « "2. Airports by Country" « std::endl;
00542   std::cout « "3. Airports by City" « std::endl;
00543   std::cout « "0. Back to Main Menu" « std::endl;
00544   std::cout « "-------------------------------------------------" « std::endl;
00545
00546   std::cin » flag;
00547   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00548
00549   while (std::cin.fail())
00550   {
00551     std::cin.clear();
00552     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00553     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00554     menuFlights();
00555   }
00556
00557   switch (flag)
00558   {
00559   case 1:
00560     std::cout « "All Airports: " « quantityAirports(airports) « std::endl;
00561     std::cout « "-------------------------------------------------"
00562               « std::endl;
00563     std::cout « "Press any key to continue..." « std::endl;
00564     std::cin.ignore();
00565     std::cin.get();
00566     menuQuantity();
00567     break;
00568   case 2:
00569     std::cout « "Type Country Name: " « std::endl;
00570     std::getline(std::cin, arg);
00571
00572     while (countriesHash.find(arg) == countriesHash.end())
00573     {
00574       std::cout « "Country not found" « std::endl;
00575       std::cout « "Type Country Name: " « std::endl;
00576       std::getline(std::cin, arg);
00577     }
00578
00579     std::cout « "Airports in " « arg « ": "
00580               « quantityAirportsCountry(airports, arg) « std::endl;
00581     std::cout « "-------------------------------------------------"
00582               « std::endl;
00583     std::cout « "Press any key to continue..." « std::endl;
00584     std::cin.ignore();
00585     std::cin.get();
00586     menuQuantity();
00587     break;
00588   case 3:
00589     std::cout « "Type City Name: " « std::endl;
00590     std::getline(std::cin, arg);
00591
00592     while (citiesHash.find(arg) == citiesHash.end())
00593     {
00594       std::cout « "City not found" « std::endl;
00595       std::cout « "Type City Name: " « std::endl;
00596       std::getline(std::cin, arg);
00597     }
00598
00599     std::cout « "Type Country Name: " « std::endl;
00600     std::getline(std::cin, arg2);
00601
00602     while (countriesHash.find(arg2) == countriesHash.end())
00603     {
00604       std::cout « "Country not found" « std::endl;
00605       std::cout « "Type Country Name: " « std::endl;
00606       std::getline(std::cin, arg2);
00607     }
00608
00609     std::cout « "Airports in " « arg « ", " « arg2 « ": "
00610               « quantityAirportsCity(airports, arg) « std::endl;
```

```
00611     std::cout « "-------------------------------------------------"
00612               « std::endl;
00613     std::cout « "Press any key to continue..." « std::endl;
00614     std::cin.ignore();
00615     std::cin.get();
00616     menuQuantity();
00617     break;
00618   case 0:
00619     menuQuantity();
00620     break;
00621   default:
00622     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00623     menuAirports();
00624     break;
00625   }
00626 }
00627
00632 void menuCountries()
00633 {
00634   int flag;
00635   std::string arg;
00636
00637   system("clear");
00638   std::cout « "Number of Countries:" « std::endl;
00639   std::cout « "-------------------------------------------------" « std::endl;
00640
00641   std::cout « "1. All Countries" « std::endl;
00642   std::cout « "0. Back to Main Menu" « std::endl;
00643   std::cout « "-------------------------------------------------" « std::endl;
00644
00645   std::cin » flag;
00646   while (std::cin.fail())
00647   {
00648     std::cin.clear();
00649     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00650     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00651     menuFlights();
00652   }
00653
00654   switch (flag)
00655   {
00656   case 1:
00657     std::cout « "All Countries: " « countriesHash.size() « std::endl;
00658     std::cout « "-------------------------------------------------"
00659               « std::endl;
00660     std::cout « "Press any key to continue..." « std::endl;
00661     std::cin.ignore();
00662     std::cin.get();
00663     menuQuantity();
00664     break;
00665   case 0:
00666     menuQuantity();
00667     break;
00668   default:
00669     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00670     menuCountries();
00671     break;
00672   }
00673 }
00674
00679 void menuCities()
00680 {
00681   int flag;
00682   std::string arg;
00683
00684   system("clear");
00685   std::cout « "Number of Cities:" « std::endl;
00686   std::cout « "-------------------------------------------------" « std::endl;
00687
00688   std::cout « "1. All Cities" « std::endl;
00689   std::cout « "2. Cities by Country" « std::endl;
00690   std::cout « "0. Back to Main Menu" « std::endl;
00691   std::cout « "-------------------------------------------------" « std::endl;
00692
00693   std::cin » flag;
00694   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00695
00696   while (std::cin.fail())
00697   {
00698     std::cin.clear();
00699     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00700     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00701     menuFlights();
00702   }
00703
00704   switch (flag)
00705   {
```

```
00706   case 1:
00707     std::cout « "All Cities: " « citiesHash.size() « std::endl;
00708     std::cout « "-------------------------------------------------"
00709               « std::endl;
00710     std::cout « "Press any key to continue..." « std::endl;
00711     std::cin.ignore();
00712     std::cin.get();
00713     menuQuantity();
00714     break;
00715   case 2:
00716     std::cout « "Type Country Name: " « std::endl;
00717     std::getline(std::cin, arg);
00718 
00719     while (countriesHash.find(arg) == countriesHash.end())
00720     {
00721       std::cout « "Country not found" « std::endl;
00722       std::cout « "Type Country Name: " « std::endl;
00723       std::getline(std::cin, arg);
00724     }
00725 
00726     std::cout « "Cities in " « arg « ": "
00727               « quantityCitiesCountry(airports, arg) « std::endl;
00728     std::cout « "-------------------------------------------------"
00729               « std::endl;
00730     std::cout « "Press any key to continue..." « std::endl;
00731     std::cin.ignore();
00732     std::cin.get();
00733     menuQuantity();
00734     break;
00735   case 0:
00736     menuQuantity();
00737     break;
00738   default:
00739     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00740     menuCities();
00741     break;
00742   }
00743 }
00744 
00749 void menuAirlines()
00750 {
00751   int flag;
00752   std::string arg;
00753 
00754   system("clear");
00755   std::cout « "Number of Airlines:" « std::endl;
00756   std::cout « "-------------------------------------------------" « std::endl;
00757 
00758   std::cout « "1. All Airlines" « std::endl;
00759   std::cout « "2. Airlines by Country" « std::endl;
00760   std::cout « "0. Back to Main Menu" « std::endl;
00761   std::cout « "-------------------------------------------------" « std::endl;
00762 
00763   std::cin » flag;
00764   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00765 
00766   while (std::cin.fail())
00767   {
00768     std::cin.clear();
00769     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00770     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00771     menuFlights();
00772   }
00773 
00774   switch (flag)
00775   {
00776   case 1:
00777     std::cout « "All Airlines: " « airlines.size()
00778               « std::endl;
00779     std::cout « "-------------------------------------------------"
00780               « std::endl;
00781     std::cout « "Press any key to continue..." « std::endl;
00782     std::cin.ignore();
00783     std::cin.get();
00784     menuQuantity();
00785     break;
00786   case 2:
00787     std::cout « "Type Country Name: " « std::endl;
00788     std::getline(std::cin, arg);
00789 
00790     while (countriesHash.find(arg) == countriesHash.end())
00791     {
00792       std::cout « "Country not found" « std::endl;
00793       std::cout « "Type Country Name: " « std::endl;
00794       std::getline(std::cin, arg);
00795     }
00796 
```

```
00797     std::cout « "Airlines in " « arg « ": " « quantityAirlinesCountry(airlines, arg)
00798               « std::endl;
00799     std::cout « "-------------------------------------------------"
00800               « std::endl;
00801     std::cout « "Press any key to continue..." « std::endl;
00802     std::cin.ignore();
00803     std::cin.get();
00804     menuQuantity();
00805     break;
00806   case 0:
00807     menuQuantity();
00808     break;
00809   default:
00810     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00811     menuAirlines();
00812     break;
00813   }
00814 }
00815
00820 void menuFlights()
00821 {
00822   int flag;
00823   std::string arg;
00824   std::string arg2;
00825
00826   system("clear");
00827   std::cout « "Number of flights:" « std::endl;
00828   std::cout « "-------------------------------------------------" « std::endl;
00829
00830   std::cout « "1. All flights" « std::endl;
00831   std::cout « "2. Flights by Origin Airport" « std::endl;
00832   std::cout « "3. Flights by Origin Country" « std::endl;
00833   std::cout « "4. Flights by City" « std::endl;
00834   std::cout « "5. Flights by Airline" « std::endl;
00835   std::cout « "0. Back to Main Menu" « std::endl;
00836   std::cout « "-------------------------------------------------" « std::endl;
00837
00838   std::cin » flag;
00839   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00840
00841   while (std::cin.fail())
00842   {
00843     std::cin.clear();
00844     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00845     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00846     menuFlights();
00847   }
00848
00849   switch (flag)
00850   {
00851   case 1:
00852     std::cout « "All flights: " « quantityFlights(airports) « std::endl;
00853     std::cout « "-------------------------------------------------"
00854               « std::endl;
00855     std::cout « "Press any key to continue..." « std::endl;
00856     std::cin.ignore();
00857     std::cin.get();
00858     menuQuantity();
00859     break;
00860   case 2:
00861     std::cout « "Type Origin Airport Code: " « std::endl;
00862     std::cin » arg;
00863
00864     while (airportsHash.find(arg) == airportsHash.end())
00865     {
00866       std::cout « "Airport not found" « std::endl;
00867       std::cout « "Type Origin Airport Code: " « std::endl;
00868       std::cin » arg;
00869     }
00870
00871     std::cout « "Flights in " « arg « ": "
00872               « quantityFlightsAirport(airports, arg) « std::endl;
00873     std::cout « "-------------------------------------------------"
00874               « std::endl;
00875     std::cout « "Press any key to continue..." « std::endl;
00876     std::cin.ignore();
00877     std::cin.get();
00878     menuQuantity();
00879     break;
00880   case 3:
00881     std::cout « "Type Origin Country Code: " « std::endl;
00882     std::getline(std::cin, arg);
00883
00884     while (countriesHash.find(arg) == countriesHash.end())
00885     {
00886       std::cout « "Country not found" « std::endl;
00887       std::cout « "Type Origin Country Code: " « std::endl;
```

```
00888       std::getline(std::cin, arg);
00889     }
00890
00891     std::cout « "Flights in " « arg « ": "
00892               « quantityFlightsCountry(airports, arg) « std::endl;
00893     std::cout « "-------------------------------------------------"
00894               « std::endl;
00895     std::cout « "Press any key to continue..." « std::endl;
00896     std::cin.ignore();
00897     std::cin.get();
00898     menuQuantity();
00899     break;
00900   case 4:
00901     std::cout « "Type City Name: " « std::endl;
00902     std::getline(std::cin, arg);
00903
00904     while (citiesHash.find(arg) == citiesHash.end())
00905     {
00906       std::cout « "City not found" « std::endl;
00907       std::cout « "Type City Name: " « std::endl;
00908       std::getline(std::cin, arg);
00909     }
00910
00911     std::cout « "Type Country Name: " « std::endl;
00912     std::getline(std::cin, arg2);
00913
00914     while (countriesHash.find(arg2) == countriesHash.end())
00915     {
00916       std::cout « "Country not found" « std::endl;
00917       std::cout « "Type Country Name: " « std::endl;
00918       std::getline(std::cin, arg2);
00919     }
00920
00921     std::cout « "Flights in " « arg « ": "
00922               « quantityFlightsCity(airports, arg) « std::endl;
00923     std::cout « "-------------------------------------------------"
00924               « std::endl;
00925     std::cout « "Press any key to continue..." « std::endl;
00926     std::cin.ignore();
00927     std::cin.get();
00928     menuQuantity();
00929     break;
00930   case 5:
00931     std::cout « "Type Airline Code: " « std::endl;
00932     std::cin » arg;
00933     std::cout « "Flights in " « arg « ": "
00934               « quantityFlightsAirline(airports, arg) « std::endl;
00935     std::cout « "-------------------------------------------------"
00936               « std::endl;
00937     std::cout « "Press any key to continue..." « std::endl;
00938     std::cin.ignore();
00939     std::cin.get();
00940     menuQuantity();
00941     break;
00942   case 0:
00943     menuQuantity();
00944     break;
00945   default:
00946     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00947     menuFlights();
00948     break;
00949   }
00950 }
00951
00956 void menuDestination()
00957 {
00958   int flag;
00959   std::string arg;
00960   int stop;
00961
00962   system("clear");
00963   std::cout « "Number of Destinations:" « std::endl;
00964   std::cout « "-------------------------------------------------" « std::endl;
00965
00966   std::cout « "1. Unlimited Stops (by airport)" « std::endl;
00967   std::cout « "2. Limited Stops" « std::endl;
00968   std::cout « "3. Max destinations" « std::endl;
00969   std::cout « "0. Back to Main Menu" « std::endl;
00970   std::cout « "-------------------------------------------------" « std::endl;
00971
00972   std::cin » flag;
00973   while (std::cin.fail())
00974   {
00975     std::cin.clear();
00976     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00977     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00978     menuDestination();
```

```
00979   }
00980
00981   switch (flag)
00982   {
00983   case 1:
00984     std::cout « "Type Airport Code: " « std::endl;
00985     std::cin » arg;
00986
00987     if (airportsHash.find(arg) == airportsHash.end())
00988     {
00989       std::cout « "Airport not found" « std::endl;
00990       std::cout « "-------------------------------------------------"
00991                 « std::endl;
00992       std::cout « "Press any key to continue..." « std::endl;
00993       std::cin.ignore();
00994       std::cin.get();
00995       menuDestination();
00996     }
00997
00998     std::cout « "Destination in " « arg « ": "
00999               « quantityDestinationsAirport(airports, arg) « std::endl;
01000     std::cout « "-------------------------------------------------"
01001               « std::endl;
01002     std::cout « "Press any key to continue..." « std::endl;
01003     std::cin.ignore();
01004     std::cin.get();
01005     menuQuantity();
01006     break;
01007   case 2:
01008     std::cout « "Type Airport Code: " « std::endl;
01009     std::cin » arg;
01010
01011     if (airportsHash.find(arg) == airportsHash.end())
01012     {
01013       std::cout « "Airport not found" « std::endl;
01014       std::cout « "-------------------------------------------------"
01015                 « std::endl;
01016       std::cout « "Press any key to continue..." « std::endl;
01017       std::cin.ignore();
01018       std::cin.get();
01019       menuDestination();
01020     }
01021
01022     std::cout « "Type Stops Number: " « std::endl;
01023     std::cin » stop;
01024     while (std::cin.fail())
01025     {
01026       std::cin.clear();
01027       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
01028       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
01029       Menu("");
01030     }
01031     std::cout « "Number of Countries: "
01032               « quantityDestinationLimitedStop(airports, arg, stop)
01033               « std::endl;
01034     std::cout « "-------------------------------------------------"
01035               « std::endl;
01036     std::cout « "Press any key to continue..." « std::endl;
01037     std::cin.ignore();
01038     std::cin.get();
01039     menuDestination();
01040   case 3:
01041     std::cout « "Max number of flights: " « quantityDestinationMax(airports)
01042               « " stops" « std::endl;
01043     std::cout « "-------------------------------------------------"
01044               « std::endl;
01045     std::cout « "Press any key to continue..." « std::endl;
01046     std::cin.ignore();
01047     std::cin.get();
01048     menuDestination();
01049   case 0:
01050     Menu("");
01051     break;
01052   default:
01053     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
01054     menuDestination();
01055     break;
01056   }
01057 }
```

4.23 src/Menu.h File Reference

```
#include "database/dbairport.h"
#include "database/read.h"
#include <iostream>
#include <limits>
```

Functions

- void Menu (std::string folder)
  Menu function to display the main menu.
- void menuQuantity ()
  Function to handle quantity-related menu options.
- void menuListing ()
  Function to handle listing-related menu options.
- void menuAirports ()
  Function Menu Airports.
- void menuCountries ()
  Function Menu Countries.
- void menuCities ()
  Function Menu Cities.
- void menuAirlines ()
  Function Menu Airlines.
- void menuFlights ()
  Function Menu Flights.
- void menuDestination ()
  Function Menu Destination.
- void bestFlights ()
  Function to find best flights based on the user input.
- int selectType (std::string arg)
  Function to handle best flights menu options.
- std::string typeAirport (std::string type, int flag)
  Function to input airport code based on the type and flag.
- pair< std::string, std::string > typeCity (std::string type, int flag)
  Function to input city details based on the type and flag.
- pair< double, double > typeCoordinates (std::string type, int flag)
  Function to input coordinates based on the type and flag.
- vector< string > filterAirplanes ()
  Function to filter airplanes on the user input.

4.23.1 Function Documentation

4.23.1.1 bestFlights()

```
void bestFlights ( )
```

Function to find best flights based on the user input.

Returns

void

Definition at line 383 of file Menu.cpp.

```
00384 {
00385   vector<string> airplanes = filterAirplanes();
00386
00387   std::string airportOrig;
00388   std::string airportDest;
00389
00390   std::pair<std::string, std::string> cityOrig;
00391   std::pair<std::string, std::string> cityDest;
00392
00393   std::pair<double, double> cordOrig;
00394   std::pair<double, double> cordDest;
00395
00396   int maxDist;
00397   int flagOrigin = selectType("origin");
00398
00399   switch (flagOrigin)
00400   {
00401   case (1):
00402     airportOrig = typeAirport("origin", flagOrigin);
00403     break;
00404   case (2):
00405     cityOrig = typeCity("origin", flagOrigin);
00406     break;
00407   case (3):
00408     cordOrig = typeCoordinates("origin", flagOrigin);
00409     std::cout « "Type Max Distance in (km): " « std::endl;
00410     std::cin » maxDist;
00411     while (std::cin.fail())
00412     {
00413       std::cin.clear();
00414       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00415       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00416       Menu("");
00417     }
00418     break;
00419   case (0):
00420     bestFlights();
00421     break;
00422   default:
00423     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00424     bestFlights();
00425     break;
00426   }
00427
00428   int flagDest = selectType("destination");
00429   switch (flagDest)
00430   {
00431   case (1):
00432     airportDest = typeAirport("destination", flagDest);
00433     break;
00434   case (2):
00435     cityDest = typeCity("destination", flagDest);
00436     break;
00437   case (3):
00438     cordDest = typeCoordinates("destination", flagDest);
00439     std::cout « "Type Max Distance in (km): " « std::endl;
00440     std::cin » maxDist;
00441     while (std::cin.fail())
00442     {
00443       std::cin.clear();
00444       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00445       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00446       Menu("");
00447     }
00448     break;
00449   case (0):
00450     bestFlights();
00451     break;
00452   default:
00453     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00454     bestFlights();
00455     break;
00456   }
00457
00458   switch (flagOrigin)
00459   {
00460   case (1):
00461     switch (flagDest)
00462     {
00463       {
```

```
00464       case (1): // airport to airport
00465         findBestFlights(airports, airportOrig, airportDest, airplanes);
00466         break;
00467       case (2): // airport to city
00468         findBestFlights(airports, cityDest.second, cityDest.first, airportOrig,
00469                         1, airplanes);
00470         break;
00471       case (3): // airport to coordinates
00472         findBestFlights(airports, airportOrig, cordDest.first, cordDest.second,
00473                         maxDist, 1, airplanes);
00474         break;
00475       }
00476     }
00477     break;
00478   case (2):
00479     switch (flagDest)
00480     {
00481       {
00482       case (1): // city to airport
00483         findBestFlights(airports, cityOrig.second, cityOrig.first, airportDest,
00484                         0, airplanes);
00485         break;
00486       case (2): // city to city
00487         findBestFlights(airports, cityOrig.second, cityOrig.first,
00488                         cityDest.second, cityDest.first, airplanes);
00489         break;
00490       case (3): // city to coordinates
00491         findBestFlights(airports, cityOrig.second, cityOrig.first,
00492                         cordDest.first, cordDest.second, maxDist, 0, airplanes);
00493         break;
00494       }
00495     }
00496     break;
00497   case (3):
00498     switch (flagDest)
00499     {
00500       {
00501       case (1): // coordinates to airport
00502         findBestFlights(airports, airportDest, cordOrig.first, cordOrig.second,
00503                         maxDist, 0, airplanes);
00504         break;
00505       case (2): // coordinates to city
00506         findBestFlights(airports, cityDest.second, cityDest.first,
00507                         cordOrig.first, cordOrig.second, maxDist, 1, airplanes);
00508         break;
00509       case (3): // coordinates to coordinates
00510         findBestFlights(airports, cordOrig.first, cordOrig.second,
00511                         cordDest.first, cordDest.second, maxDist, airplanes);
00512         break;
00513       }
00514     }
00515     break;
00516   }
00517
00518   std::cout « "-------------------------------------------------" « std::endl;
00519
00520   std::cout « "Press any key to continue..." « std::endl;
00521   std::cin.ignore();
00522   std::cin.get();
00523   bestFlights();
00524 }
```

4.23.1.2 filterAirplanes()

```
vector< string > filterAirplanes ( )
```

Function to filter airplanes on the user input.

Returns

vector<string> The vector of airplanes to filter

Definition at line 322 of file Menu.cpp.

```
00323 {
00324   vector<string> airplanes;
00325   string line;
00326   int flag;
```

```
00327
00328   system("clear");
00329   std::cout « "Bests flights: " « std::endl;
00330   std::cout « "-------------------------------------------------" « std::endl;
00331
00332   std::cout « "1. Filter by airplanes" « std::endl;
00333   std::cout « "2. Without filter" « std::endl;
00334   std::cout « "0. Back to Main Menu" « std::endl;
00335   std::cout « "-------------------------------------------------" « std::endl;
00336
00337   std::cin » flag;
00338   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00339
00340   while (std::cin.fail())
00341   {
00342     std::cout « "Erro na leitura" « std::endl;
00343     std::cin.clear();
00344     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00345     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00346     filterAirplanes();
00347   }
00348
00349   switch (flag)
00350   {
00351   case (1):
00352   {
00353     std::cout « "Type Airplane Name: " « std::endl;
00354     std::getline(std::cin, line);
00355
00356     std::istringstream iss(line);
00357
00358     string airplane;
00359
00360     while (iss » airplane)
00361     {
00362       airplanes.push_back(airplane);
00363     }
00364     break;
00365   }
00366   case (2):
00367     break;
00368   case (0):
00369     Menu("");
00370     break;
00371   default:
00372     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00373     filterAirplanes();
00374     break;
00375   }
00376   return airplanes;
00377 }
```

4.23.1.3 Menu()

```
void Menu (
            std::string folder )
```

Menu function to display the main menu.

Parameters

folder	The folder name to read the data from.

Returns

void

Definition at line 13 of file Menu.cpp.

```
00014 {
00015   int flag;
00016
00017   if (!folder.empty())
00018   {
```

```
00019     airports = readFlights(folder);
00020     airlines = readAirlines(folder);
00021   }
00022
00023   system("clear");
00024   std::cout « "Welcome to Travel Management:" « std::endl;
00025   std::cout « "-----------------------------------------------" « std::endl;
00026   std::cout « "1. Quantity calculation" « std::endl;
00027   std::cout « "2. Listing" « std::endl;
00028   std::cout « "3. Bests flights" « std::endl;
00029   std::cout « "0. Exit" « std::endl;
00030   std::cout « "-----------------------------------------------" « std::endl;
00031
00032   std::cin » flag;
00033   while (std::cin.fail())
00034   {
00035     std::cin.clear();
00036     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00037     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00038     Menu("");
00039   }
00040
00041   switch (flag)
00042   {
00043   case 1:
00044     menuQuantity();
00045     break;
00046   case 2:
00047     menuListing();
00048     break;
00049   case 3:
00050     bestFlights();
00051     break;
00052   case 0:
00053     exit(0);
00054     break;
00055   default:
00056     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00057     Menu("");
00058     break;
00059   }
00060 }
```

4.23.1.4 menuAirlines()

```
void menuAirlines ( )
```

Function Menu Airlines.

Returns

void

Definition at line 749 of file Menu.cpp.

```
00750 {
00751   int flag;
00752   std::string arg;
00753
00754   system("clear");
00755   std::cout « "Number of Airlines:" « std::endl;
00756   std::cout « "-----------------------------------------------" « std::endl;
00757
00758   std::cout « "1. All Airlines" « std::endl;
00759   std::cout « "2. Airlines by Country" « std::endl;
00760   std::cout « "0. Back to Main Menu" « std::endl;
00761   std::cout « "-----------------------------------------------" « std::endl;
00762
00763   std::cin » flag;
00764   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00765
00766   while (std::cin.fail())
00767   {
00768     std::cin.clear();
00769     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00770     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00771     menuFlights();
00772   }
00773
```

```
00774   switch (flag)
00775   {
00776   case 1:
00777     std::cout « "All Airlines: " « airlines.size()
00778               « std::endl;
00779     std::cout « "-------------------------------------------------"
00780               « std::endl;
00781     std::cout « "Press any key to continue..." « std::endl;
00782     std::cin.ignore();
00783     std::cin.get();
00784     menuQuantity();
00785     break;
00786   case 2:
00787     std::cout « "Type Country Name: " « std::endl;
00788     std::getline(std::cin, arg);
00789
00790     while (countriesHash.find(arg) == countriesHash.end())
00791     {
00792       std::cout « "Country not found" « std::endl;
00793       std::cout « "Type Country Name: " « std::endl;
00794       std::getline(std::cin, arg);
00795     }
00796
00797     std::cout « "Airlines in " « arg « ": " « quantityAirlinesCountry(airlines, arg)
00798               « std::endl;
00799     std::cout « "-------------------------------------------------"
00800               « std::endl;
00801     std::cout « "Press any key to continue..." « std::endl;
00802     std::cin.ignore();
00803     std::cin.get();
00804     menuQuantity();
00805     break;
00806   case 0:
00807     menuQuantity();
00808     break;
00809   default:
00810     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00811     menuAirlines();
00812     break;
00813   }
00814 }
```

4.23.1.5 menuAirports()

```
void menuAirports ( )
```

Function Menu Airports.

Returns

void

Definition at line 530 of file Menu.cpp.

```
00531 {
00532   int flag;
00533   std::string arg;
00534   std::string arg2;
00535
00536   system("clear");
00537   std::cout « "Number of Airports:" « std::endl;
00538   std::cout « "-------------------------------------------------" « std::endl;
00539
00540   std::cout « "1. All Airports" « std::endl;
00541   std::cout « "2. Airports by Country" « std::endl;
00542   std::cout « "3. Airports by City" « std::endl;
00543   std::cout « "0. Back to Main Menu" « std::endl;
00544   std::cout « "-------------------------------------------------" « std::endl;
00545
00546   std::cin » flag;
00547   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00548
00549   while (std::cin.fail())
00550   {
00551     std::cin.clear();
00552     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00553     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00554     menuFlights();
```

```
00555   }
00556
00557   switch (flag)
00558   {
00559   case 1:
00560     std::cout « "All Airports: " « quantityAirports(airports) « std::endl;
00561     std::cout « "-------------------------------------------------"
00562               « std::endl;
00563     std::cout « "Press any key to continue..." « std::endl;
00564     std::cin.ignore();
00565     std::cin.get();
00566     menuQuantity();
00567     break;
00568   case 2:
00569     std::cout « "Type Country Name: " « std::endl;
00570     std::getline(std::cin, arg);
00571
00572     while (countriesHash.find(arg) == countriesHash.end())
00573     {
00574       std::cout « "Country not found" « std::endl;
00575       std::cout « "Type Country Name: " « std::endl;
00576       std::getline(std::cin, arg);
00577     }
00578
00579     std::cout « "Airports in " « arg « ": "
00580               « quantityAirportsCountry(airports, arg) « std::endl;
00581     std::cout « "-------------------------------------------------"
00582               « std::endl;
00583     std::cout « "Press any key to continue..." « std::endl;
00584     std::cin.ignore();
00585     std::cin.get();
00586     menuQuantity();
00587     break;
00588   case 3:
00589     std::cout « "Type City Name: " « std::endl;
00590     std::getline(std::cin, arg);
00591
00592     while (citiesHash.find(arg) == citiesHash.end())
00593     {
00594       std::cout « "City not found" « std::endl;
00595       std::cout « "Type City Name: " « std::endl;
00596       std::getline(std::cin, arg);
00597     }
00598
00599     std::cout « "Type Country Name: " « std::endl;
00600     std::getline(std::cin, arg2);
00601
00602     while (countriesHash.find(arg2) == countriesHash.end())
00603     {
00604       std::cout « "Country not found" « std::endl;
00605       std::cout « "Type Country Name: " « std::endl;
00606       std::getline(std::cin, arg2);
00607     }
00608
00609     std::cout « "Airports in " « arg « ", " « arg2 « ": "
00610               « quantityAirportsCity(airports, arg) « std::endl;
00611     std::cout « "-------------------------------------------------"
00612               « std::endl;
00613     std::cout « "Press any key to continue..." « std::endl;
00614     std::cin.ignore();
00615     std::cin.get();
00616     menuQuantity();
00617     break;
00618   case 0:
00619     menuQuantity();
00620     break;
00621   default:
00622     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00623     menuAirports();
00624     break;
00625   }
00626 }
```

4.23.1.6 menuCities()

```
void menuCities ( )
```

Function Menu Cities.

Returns

void

Definition at line 679 of file Menu.cpp.

```
00680 {
00681   int flag;
00682   std::string arg;
00683
00684   system("clear");
00685   std::cout « "Number of Cities:" « std::endl;
00686   std::cout « "-------------------------------------------------" « std::endl;
00687
00688   std::cout « "1. All Cities" « std::endl;
00689   std::cout « "2. Cities by Country" « std::endl;
00690   std::cout « "0. Back to Main Menu" « std::endl;
00691   std::cout « "-------------------------------------------------" « std::endl;
00692
00693   std::cin » flag;
00694   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00695
00696   while (std::cin.fail())
00697   {
00698     std::cin.clear();
00699     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00700     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00701     menuFlights();
00702   }
00703
00704   switch (flag)
00705   {
00706   case 1:
00707     std::cout « "All Cities: " « citiesHash.size() « std::endl;
00708     std::cout « "-------------------------------------------------"
00709               « std::endl;
00710     std::cout « "Press any key to continue..." « std::endl;
00711     std::cin.ignore();
00712     std::cin.get();
00713     menuQuantity();
00714     break;
00715   case 2:
00716     std::cout « "Type Country Name: " « std::endl;
00717     std::getline(std::cin, arg);
00718
00719     while (countriesHash.find(arg) == countriesHash.end())
00720     {
00721       std::cout « "Country not found" « std::endl;
00722       std::cout « "Type Country Name: " « std::endl;
00723       std::getline(std::cin, arg);
00724     }
00725
00726     std::cout « "Cities in " « arg « ": "
00727               « quantityCitiesCountry(airports, arg) « std::endl;
00728     std::cout « "-------------------------------------------------"
00729               « std::endl;
00730     std::cout « "Press any key to continue..." « std::endl;
00731     std::cin.ignore();
00732     std::cin.get();
00733     menuQuantity();
00734     break;
00735   case 0:
00736     menuQuantity();
00737     break;
00738   default:
00739     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00740     menuCities();
00741     break;
00742   }
00743 }
```

4.23.1.7 menuCountries()

```
void menuCountries ( )
```

Function Menu Countries.

Returns

void

Definition at line 632 of file Menu.cpp.

```
00633 {
00634   int flag;
00635   std::string arg;
00636
00637   system("clear");
00638   std::cout « "Number of Countries:" « std::endl;
00639   std::cout « "-------------------------------------------------" « std::endl;
00640
00641   std::cout « "1. All Countries" « std::endl;
00642   std::cout « "0. Back to Main Menu" « std::endl;
00643   std::cout « "-------------------------------------------------" « std::endl;
00644
00645   std::cin » flag;
00646   while (std::cin.fail())
00647   {
00648     std::cin.clear();
00649     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00650     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00651     menuFlights();
00652   }
00653
00654   switch (flag)
00655   {
00656   case 1:
00657     std::cout « "All Countries: " « countriesHash.size() « std::endl;
00658     std::cout « "-------------------------------------------------"
00659               « std::endl;
00660     std::cout « "Press any key to continue..." « std::endl;
00661     std::cin.ignore();
00662     std::cin.get();
00663     menuQuantity();
00664     break;
00665   case 0:
00666     menuQuantity();
00667     break;
00668   default:
00669     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00670     menuCountries();
00671     break;
00672   }
00673 }
```

4.23.1.8 menuDestination()

```
void menuDestination ( )
```

Function Menu Destination.

Returns

void

Definition at line 956 of file Menu.cpp.

```
00957 {
00958   int flag;
00959   std::string arg;
00960   int stop;
00961
00962   system("clear");
00963   std::cout « "Number of Destinations:" « std::endl;
00964   std::cout « "-------------------------------------------------" « std::endl;
00965
00966   std::cout « "1. Unlimited Stops (by airport)" « std::endl;
00967   std::cout « "2. Limited Stops" « std::endl;
00968   std::cout « "3. Max destinations" « std::endl;
00969   std::cout « "0. Back to Main Menu" « std::endl;
00970   std::cout « "-------------------------------------------------" « std::endl;
00971
00972   std::cin » flag;
00973   while (std::cin.fail())
00974   {
```

```
00975     std::cin.clear();
00976     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00977     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00978     menuDestination();
00979   }
00980
00981   switch (flag)
00982   {
00983   case 1:
00984     std::cout « "Type Airport Code: " « std::endl;
00985     std::cin » arg;
00986
00987     if (airportsHash.find(arg) == airportsHash.end())
00988     {
00989       std::cout « "Airport not found" « std::endl;
00990       std::cout « "-------------------------------------------------"
00991                 « std::endl;
00992       std::cout « "Press any key to continue..." « std::endl;
00993       std::cin.ignore();
00994       std::cin.get();
00995       menuDestination();
00996     }
00997
00998     std::cout « "Destination in " « arg « ": "
00999               « quantityDestinationsAirport(airports, arg) « std::endl;
01000     std::cout « "-------------------------------------------------"
01001               « std::endl;
01002     std::cout « "Press any key to continue..." « std::endl;
01003     std::cin.ignore();
01004     std::cin.get();
01005     menuQuantity();
01006     break;
01007   case 2:
01008     std::cout « "Type Airport Code: " « std::endl;
01009     std::cin » arg;
01010
01011     if (airportsHash.find(arg) == airportsHash.end())
01012     {
01013       std::cout « "Airport not found" « std::endl;
01014       std::cout « "-------------------------------------------------"
01015                 « std::endl;
01016       std::cout « "Press any key to continue..." « std::endl;
01017       std::cin.ignore();
01018       std::cin.get();
01019       menuDestination();
01020     }
01021
01022     std::cout « "Type Stops Number: " « std::endl;
01023     std::cin » stop;
01024     while (std::cin.fail())
01025     {
01026       std::cin.clear();
01027       std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
01028       std::cout « "Invalid input. Please enter a valid number: " « std::endl;
01029       Menu("");
01030     }
01031     std::cout « "Number of Countries: "
01032               « quantityDestinationLimitedStop(airports, arg, stop)
01033               « std::endl;
01034     std::cout « "-------------------------------------------------"
01035               « std::endl;
01036     std::cout « "Press any key to continue..." « std::endl;
01037     std::cin.ignore();
01038     std::cin.get();
01039     menuDestination();
01040   case 3:
01041     std::cout « "Max number of flights: " « quantityDestinationMax(airports)
01042               « " stops" « std::endl;
01043     std::cout « "-------------------------------------------------"
01044               « std::endl;
01045     std::cout « "Press any key to continue..." « std::endl;
01046     std::cin.ignore();
01047     std::cin.get();
01048     menuDestination();
01049   case 0:
01050     Menu("");
01051     break;
01052   default:
01053     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
01054     menuDestination();
01055     break;
01056   }
01057 }
```

4.23.1.9 menuFlights()

```
void menuFlights ( )
```

Function Menu Flights.

Returns

void

Definition at line 820 of file Menu.cpp.

```
00821 {
00822   int flag;
00823   std::string arg;
00824   std::string arg2;
00825 
00826   system("clear");
00827   std::cout « "Number of flights:" « std::endl;
00828   std::cout « "-------------------------------------------------" « std::endl;
00829 
00830   std::cout « "1. All flights" « std::endl;
00831   std::cout « "2. Flights by Origin Airport" « std::endl;
00832   std::cout « "3. Flights by Origin Country" « std::endl;
00833   std::cout « "4. Flights by City" « std::endl;
00834   std::cout « "5. Flights by Airline" « std::endl;
00835   std::cout « "0. Back to Main Menu" « std::endl;
00836   std::cout « "-------------------------------------------------" « std::endl;
00837 
00838   std::cin » flag;
00839   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00840 
00841   while (std::cin.fail())
00842   {
00843     std::cin.clear();
00844     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00845     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00846     menuFlights();
00847   }
00848 
00849   switch (flag)
00850   {
00851   case 1:
00852     std::cout « "All flights: " « quantityFlights(airports) « std::endl;
00853     std::cout « "-------------------------------------------------"
00854               « std::endl;
00855     std::cout « "Press any key to continue..." « std::endl;
00856     std::cin.ignore();
00857     std::cin.get();
00858     menuQuantity();
00859     break;
00860   case 2:
00861     std::cout « "Type Origin Airport Code: " « std::endl;
00862     std::cin » arg;
00863 
00864     while (airportsHash.find(arg) == airportsHash.end())
00865     {
00866       std::cout « "Airport not found" « std::endl;
00867       std::cout « "Type Origin Airport Code: " « std::endl;
00868       std::cin » arg;
00869     }
00870 
00871     std::cout « "Flights in " « arg « ": "
00872               « quantityFlightsAirport(airports, arg) « std::endl;
00873     std::cout « "-------------------------------------------------"
00874               « std::endl;
00875     std::cout « "Press any key to continue..." « std::endl;
00876     std::cin.ignore();
00877     std::cin.get();
00878     menuQuantity();
00879     break;
00880   case 3:
00881     std::cout « "Type Origin Country Code: " « std::endl;
00882     std::getline(std::cin, arg);
00883 
00884     while (countriesHash.find(arg) == countriesHash.end())
00885     {
00886       std::cout « "Country not found" « std::endl;
00887       std::cout « "Type Origin Country Code: " « std::endl;
00888       std::getline(std::cin, arg);
00889     }
```

```
00890
00891     std::cout « "Flights in " « arg « ": "
00892               « quantityFlightsCountry(airports, arg) « std::endl;
00893     std::cout « "-------------------------------------------------"
00894               « std::endl;
00895     std::cout « "Press any key to continue..." « std::endl;
00896     std::cin.ignore();
00897     std::cin.get();
00898     menuQuantity();
00899     break;
00900   case 4:
00901     std::cout « "Type City Name: " « std::endl;
00902     std::getline(std::cin, arg);
00903
00904     while (citiesHash.find(arg) == citiesHash.end())
00905     {
00906       std::cout « "City not found" « std::endl;
00907       std::cout « "Type City Name: " « std::endl;
00908       std::getline(std::cin, arg);
00909     }
00910
00911     std::cout « "Type Country Name: " « std::endl;
00912     std::getline(std::cin, arg2);
00913
00914     while (countriesHash.find(arg2) == countriesHash.end())
00915     {
00916       std::cout « "Country not found" « std::endl;
00917       std::cout « "Type Country Name: " « std::endl;
00918       std::getline(std::cin, arg2);
00919     }
00920
00921     std::cout « "Flights in " « arg « ": "
00922               « quantityFlightsCity(airports, arg) « std::endl;
00923     std::cout « "-------------------------------------------------"
00924               « std::endl;
00925     std::cout « "Press any key to continue..." « std::endl;
00926     std::cin.ignore();
00927     std::cin.get();
00928     menuQuantity();
00929     break;
00930   case 5:
00931     std::cout « "Type Airline Code: " « std::endl;
00932     std::cin » arg;
00933     std::cout « "Flights in " « arg « ": "
00934               « quantityFlightsAirline(airports, arg) « std::endl;
00935     std::cout « "-------------------------------------------------"
00936               « std::endl;
00937     std::cout « "Press any key to continue..." « std::endl;
00938     std::cin.ignore();
00939     std::cin.get();
00940     menuQuantity();
00941     break;
00942   case 0:
00943     menuQuantity();
00944     break;
00945   default:
00946     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00947     menuFlights();
00948     break;
00949   }
00950 }
```

4.23.1.10 menuListing()

```
void menuListing ( )
```

Function to handle listing-related menu options.

Returns

void

Definition at line 132 of file Menu.cpp.

```
00133 {
00134   int flag;
00135   int arg;
00136
```

```
00137   system("clear");
00138   std::cout « "Listing Menu: " « std::endl;
00139   std::cout « "-------------------------------------------------" « std::endl;
00140
00141   std::cout « "1. Ranking Airports (more landings and takeoffs)" « std::endl;
00142   std::cout « "2. Connecting airports" « std::endl;
00143   std::cout « "0. Back to Main Menu" « std::endl;
00144   std::cout « "-------------------------------------------------" « std::endl;
00145
00146   std::cin » flag;
00147   while (std::cin.fail())
00148   {
00149     std::cin.clear();
00150     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00151     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00152     menuListing();
00153   }
00154
00155   switch (flag)
00156   {
00157   case (1):
00158     system("clear");
00159     std::cout « "Type Airports Number: " « std::endl;
00160     std::cin » arg;
00161     std::cout « "-------------------------------------------------"
00162               « std::endl;
00163     std::cout « "Ranking Airports: " « std::endl;
00164     std::cout « "-------------------------------------------------"
00165               « std::endl;
00166     rankingAirports(airports, arg);
00167     std::cout « "-------------------------------------------------"
00168               « std::endl;
00169     std::cout « "Press any key to continue..." « std::endl;
00170     std::cin.ignore();
00171     std::cin.get();
00172     menuListing();
00173     break;
00174   case (2):
00175     system("clear");
00176     std::cout « "Connecting airports: " « std::endl;
00177     std::cout « "-------------------------------------------------"
00178               « std::endl;
00179     // getArticulations(airports);
00180     findArticulationPoints(airports);
00181     // connectedComponents(airports);
00182     std::cout « "-------------------------------------------------"
00183               « std::endl;
00184     std::cout « "Press any key to continue..." « std::endl;
00185     std::cin.ignore();
00186     std::cin.get();
00187     menuListing();
00188     break;
00189   case (0):
00190     Menu("");
00191     break;
00192   default:
00193     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00194     menuListing();
00195     break;
00196   }
00197 }
```

4.23.1.11 menuQuantity()

```
void menuQuantity ( )
```

Function to handle quantity-related menu options.

Returns

void

Definition at line 66 of file Menu.cpp.

```
00067 {
00068   int flag;
00069
00070   system("clear");
```

```
00071   std::cout « "Quantity Calculation Menu" « std::endl;
00072   std::cout « "-------------------------------------------------" « std::endl;
00073
00074   std::cout « "1. Number of airports" « std::endl;
00075   std::cout « "2. Number of countries" « std::endl;
00076   std::cout « "3. Number of cities" « std::endl;
00077   std::cout « "4. Number of airlines" « std::endl;
00078   std::cout « "5. Number of flights" « std::endl;
00079   std::cout « "6. Number of destinations" « std::endl;
00080   std::cout « "0. Back to Main Menu" « std::endl;
00081   std::cout « "-------------------------------------------------" « std::endl;
00082
00083   std::cin » flag;
00084   while (std::cin.fail())
00085   {
00086     std::cin.clear();
00087     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00088     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00089     menuQuantity();
00090   }
00091
00092   switch (flag)
00093   {
00094   case 1:
00095     system("clear");
00096     menuAirports();
00097     break;
00098   case 2:
00099     system("clear");
00100     menuCountries();
00101     break;
00102   case 3:
00103     system("clear");
00104     menuCities();
00105     break;
00106   case 4:
00107     system("clear");
00108     menuAirlines();
00109     break;
00110   case 5:
00111     system("clear");
00112     menuFlights();
00113     break;
00114   case 6:
00115     system("clear");
00116     menuDestination();
00117     break;
00118   case 0:
00119     Menu("");
00120     break;
00121   default:
00122     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00123     menuQuantity();
00124     break;
00125   }
00126 }
```

4.23.1.12 selectType()

```
int selectType (
            std::string arg )
```

Function to handle best flights menu options.

Parameters

arg	The type of the airport (origin or destination)

Returns

void

Definition at line 204 of file Menu.cpp.

```
00205 {
00206   int flag;
00207
00208   system("clear");
00209   std::cout « "Bests Flights: " « std::endl;
00210   std::cout « "Select your " « arg « " option:" « std::endl;
00211   std::cout « "-----------------------------------------------" « std::endl;
00212
00213   std::cout « "1. By airports" « std::endl;
00214   std::cout « "2. By cities" « std::endl;
00215   std::cout « "3. By coordinates" « std::endl;
00216   std::cout « "0. Back to Best Flights" « std::endl;
00217   std::cout « "-----------------------------------------------" « std::endl;
00218
00219   std::cin » flag;
00220   while (std::cin.fail())
00221   {
00222     std::cin.clear();
00223     std::cout « "Invalid input. Please enter a valid number: " « std::endl;
00224     selectType(arg);
00225   }
00226   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00227
00228   return flag;
00229 }
```

4.23.1.13 typeAirport()

```
std::string typeAirport (
              std::string type,
              int flag )
```

Function to input airport code based on the type and flag.

Parameters

type	The type of the airport (origin or destination)
flag	The flag to select the type of filter (airport, city or coordinates)

Returns

string The airport code

Definition at line 237 of file Menu.cpp.

```
00238 {
00239   std::string arg;
00240   switch (flag)
00241   {
00242   case (1):
00243     std::cout « "Type Code Airport of " « type « ": " « std::endl;
00244     std::cin » arg;
00245     while (airportsHash.find(arg) == airportsHash.end())
00246     {
00247       std::cout « "Airport not found" « std::endl;
00248       std::cout « "Type Code Airport of " « type « ": " « std::endl;
00249       std::cin » arg;
00250     }
00251     break;
00252   }
00253   return arg;
00254 }
```

4.23.1.14 typeCity()

```
pair< std::string, std::string > typeCity (
              std::string type,
              int flag )
```

Function to input city details based on the type and flag.

Parameters

type	The type of the airport (origin or destination)
flag	The flag to select the type of filter (airport, city or coordinates)

Returns

pair<string, string> The city name and country name

Definition at line 262 of file Menu.cpp.

```
00263 {
00264   std::string arg1;
00265   std::string arg2;
00266   switch (flag)
00267   {
00268   case (2):
00269     std::cout « "Type City Name " « type « ": " « std::endl;
00270     std::getline(std::cin, arg1);
00271
00272     while (citiesHash.find(arg1) == citiesHash.end())
00273     {
00274       std::cout « "City not found" « std::endl;
00275       std::cout « "Type City Name " « type « ": " « std::endl;
00276       std::getline(std::cin, arg1);
00277     }
00278
00279     std::cout « "Type Country Name " « type « ": " « std::endl;
00280     std::getline(std::cin, arg2);
00281
00282     while (countriesHash.find(arg2) == countriesHash.end())
00283     {
00284       std::cout « "Country not found" « std::endl;
00285       std::cout « "Type Country Name " « type « ": " « std::endl;
00286       std::getline(std::cin, arg2);
00287     }
00288
00289     break;
00290   }
00291   return make_pair(arg1, arg2);
00292 }
```

4.23.1.15 typeCoordinates()

```
pair< double, double > typeCoordinates (
            std::string type,
            int flag )
```

Function to input coordinates based on the type and flag.

Parameters

type	The type of the airport (origin or destination)
flag	The flag to select the type of filter (airport, city or coordinates)

Returns

pair<double, double> The latitude and longitude

Definition at line 300 of file Menu.cpp.

```
00301 {
00302   double arg1;
00303   double arg2;
00304   switch (flag)
```

```
00305   {
00306   case (3):
00307     std::cout « "Type Latitude of " « type « ": " « std::endl;
00308     std::cin » arg1;
00309     std::cout « "Type Longitude of " « type « ": " « std::endl;
00310     std::cin » arg2;
00311     break;
00312   }
00313   return make_pair(arg1, arg2);
00314 }
```

4.24 Menu.h

Go to the documentation of this file.

```
00001 #ifndef MENU_H
00002 #define MENU_H
00003
00004 #include "database/dbairport.h"
00005 #include "database/read.h"
00006 #include <iostream>
00007 #include <limits>
00008
00009 // Function to display menu
00010 void Menu(std::string folder);
00011 void menuQuantity();
00012 void menuListing();
00013 void menuAirports();
00014 void menuCountries();
00015 void menuCities();
00016 void menuAirlines();
00017 void menuFlights();
00018 void menuDestination();
00019
00020 // Function to handle the process of finding the best flight
00021 void bestFlights();
00022 int selectType(std::string arg);
00023 std::string typeAirport(std::string type, int flag);
00024 pair<std::string, std::string> typeCity(std::string type, int flag);
00025 pair<double, double> typeCoordinates(std::string type, int flag);
00026
00027 // Function to filter airplanes
00028 vector<string> filterAirplanes();
00029
00030 #endif
```

Index