INSTITUTO TECNOLÓGICO DE AERONÁUTICA
CURSO DE ENGENHARIA CIVIL-AERONÁUTICA

RELATÓRIO FINAL DE ESTÁGIO

ECOLE NATIONALE DE L'AVIATION CIVILE

São José dos Campos, Brasil, Outubro de 2010

FRANCO MENDONÇA YASSOYAMA

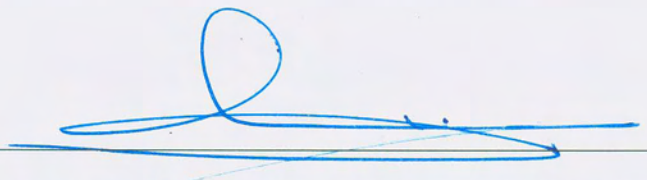
FOLHA DE APROVAÇÃO

Relatório Final de Estágio Curricular aceito em 22/10/10 pelos abaixo assinados:

FRANCO MENDONÇA YASSOYAMA

Franco Mendonça Yassoyama

p/ Félix Mora-Camino - Supervisor na ENAC



Cláudio Jorge Pinto Alves - Orientador/Supervisor no ITA

Eliseu Lucena Neto - Coordenador do Curso de Engenharia Civil-Aeronáutica

INFORMAÇÕES GERAIS

Estagiário

Nome do Aluno: Franco Mendonça Yassoyama

Curso: Engenharia Civil-Aeronáutica

Instituto / Laboratório

ENAC / LARA (Laboratoire de Recherche Opérationnelle et Automatique)

Orientador/Supervisor da Empresa

Félix Mora-Camino

Orientador/Supervisor do ITA

Prof. Cláudio Jorge Pinto Alves

Período

13/01/2010 a 21/05/2010

Contagem total de horas: 645 horas

Índice

1 INTRODUÇÃO

O relatório objetiva apresentar e descrever as atividades desenvolvidas pelo aluno Franco Mendonça Yassoyama durante o estágio supervisionado do Curso de Engenharia Civil-Aeronáutica do Instituto Tecnológico de Aeronáutica, sob a supervisão do Prof. Cláudio Jorge Pinto Alves.

2 O INSTITUTO

Fundada em 1949 em Paris e posteriormente, em 1968, mudada para Toulouse, a ENAC (Ecole Nationale de l'Aviation Civile) é uma grande universidade francesa, a qual tem por objetivo precípuo o de oferecer formação inicial e desenvolvimento de gestão dos profissionais da aviação civil do país.

Como uma "Universidade de Aviação Civil", a ENAC oferece uma ampla variedade de atividades concebidas para satisfazer as necessidades dos setores públicos e privados, tanto na França quanto em outros países. Agregam-se, no Instituto, cursos de formação, estágios, cursos especializados, pesquisas e atividades internacionais.

2.1 *O LARA*

O Laboratório de Automação e de Pesquisa Operacional (LARA, em francês), desenvolve atividades de pesquisa em dois grandes campos:

- Automação aplicada à pilotagem, orientação e gerenciamento de aviões de transporte, em colaboração com o Laboratório de Análise e de Arquitetura de Sistemas (Laboratoire d'Analyse et d'Architecture des Systèmes, LAAS) e com a Airbus. Nesse contexto, há a implementação de técnicas avançadas de

modelagem e comando por automação não-linear, e de automação simbólica (lógica Fuzzy e redes neurais). As aplicações atuais visam a segurança e a eficácia do tráfego das aeronaves, tanto no solo quanto no ar.

- Otimização da logística do transporte aéreo e pesquisa operacional aplicada ao transporte aéreo, em colaboração com o Instituto Nacional Politécnico de Toulouse (l'Institut National Polytechnique de Toulouse, INPT) e a Universidade de Toulouse II (l'Université de Toulouse II). Uma estrutura comum (**MAPTA**: **M**odelagem, **A**poio à Decisão e **P**esquisa Operacional para o **T**ransporte **A**éreo [**MARTA** : **M**odélisation, **A**ide à la Décision et **R**echerche Opérationnelle pour le **T**ransport **A**érien]) foi criada, em um acordo entre a ENAC e a Universidade de Toulouse II, para concretizar tal cooperação. As atividades de pesquisa de MAPTA se voltam para a solução de problemas operacionais ligados ao transporte aéreo.

3 ATIVIDADES DESENVOLVIDAS

3.1 *Objetivos do Estágio*

O estágio no LARA teve por objetivo adquirir conhecimento a respeito da gestão dos riscos operacionais de companhias aéreas. Ao fim do estágio, esperava-se que todo o conhecimento adquirido pudesse ser aplicado em estudos de casos de companhias aéreas nacionais. Possibilitando-se, assim, tanto a análise crítica dos resultados obtidos pela aplicação, para casos brasileiros, de ferramentas de apoio à decisão utilizadas na França, quanto o ajuste no algoritmo para que gerasse melhores resultados para os casos estudados.

3.2 *Estudos Iniciais*

Inicialmente, fui apresentado ao problema com o qual trabalharia, o ROADEF 2009 Challenge (todas as informações, descrição do problema e resultados encontram-se disponíveis em << http://challenge.roadef.org/2009/index.en.htm>>) e à linguagem de programação que seria utilizada, a do IBM ILOG Solver V6.7.

Depois, seguiram-se estudos de inteligência artificial (Nilsson, Nils J.. Principles of Artificial Intelligence), tráfego aéreo (Bazargan, Massoud. Airline Operations and Scheduling), pesquisa operacional (Hillier, Frederick S. and Lieberman, Gerald J.. Introduction to Operations Research), programação por restrições (Benhamou, Frédéric and Colmerauer, Alain. Constraint Logic Programming: Selected Research; Rossi, Francesca; van Beek, Peter and Walsh, Toby. Handbook of Constraint Programming; Roman Barták. Constraint Propagation and Backtracking-Based Search; Edward Tsang. Foundations of Constraint Satisfaction; Michael Trick. Constraint Programming; Pierre Lopez. Constraint Programming), linguagem de programação (Manuals of IBM ILOG Solver) e, por fim, um estudos das soluções apresentadas.

3.3 *Programação*

O meu obtivo foi programar, utilizando programação por restrições, parte da solução apresentada por um dos pesquisadores do LARA, Rodrigo Acuna-Agost, (juntamente com outros pesquisadores) para o problema. Um fluxograma simples que apresenta a parte que eu programei pode ser visto na Figura 01.

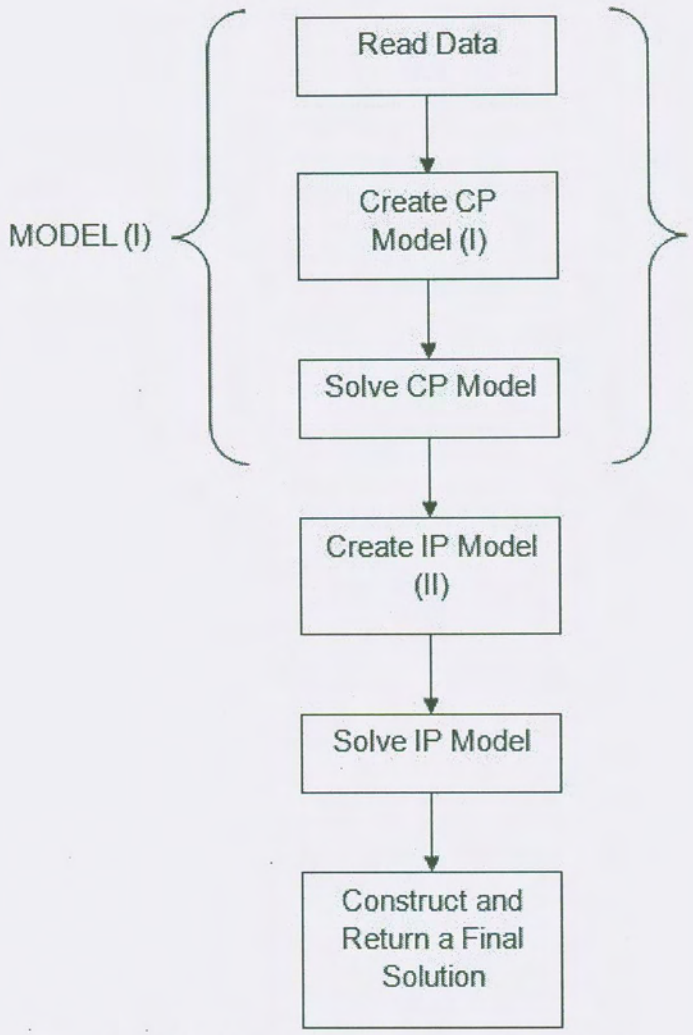


Figura 01 – Fluxograma simplificado da solução.

O algoritmo completo que foi implementado segue abaixo:

Sets:

A – aircraft, indexed by i; ("i = 0" indicates no aircraft, used to cancel flights)
P – airports, indexed by p;
F – flights, indexed by j; (add the virtual flights for "maintenance" and "breakdown")
T_p – time slots, indexed by t.

Data:

d_j – duration of flight j, j ∈ F;
tr_i – turn-round time, minimum time to prepare aircraft i for the subsequent flight, i ∈ A;
$destinationF_j$ – destination airport of flight j, j ∈ F;
$originF_j$ – origin airport of flight j, j ∈ F;
$delay_j^0$ – original delay of flight j, j ∈ F;
$arrTimeF_j^0$ – original arrival time of flight j, j ∈ F;
$depTimeF_j^0$ – original departure time of flight j, j ∈ F;
beginRW – time of the begin of the recovery window;
endRW – time of the end of the recovery window;
$capAirportDep_{pt}$ – capacity of departures for airport t in time slot t;

$capAirportArr_{pt}$ – capacity of arrivals for airport t in time slot t;
$lowerBoundTime_{pt}$ – lower bound of time slot t of airport p;
$upperBoundTime_{pt}$ – upper bound of time slot t of airport p.

Disruptions:

F^{CF} – set of canceled flights, $F^{CF} \subset F$;
F^{AB} – set of virtual flights modeling aircraft breakdowns;
F^{DF} – set of delayed flights, $F^{DF} \subset F$;
$delay_j$ – original delay of flight j, $j \in F^{DF}$;
P^D – set of airports that have a capacity disruption.

Decision Variables:

$VarAF_j$ – aircraft for flight j. $Dom(VarAF_j) = A$;
$VarDepTimeF_j$ – Departure time of flight j. $Dom(VarDepTimeF_j) = [deptimeF_j^0, \infty]$;
$VarArrTimeF_j$ – Arrival time of flight j. $Dom(VarArrTimeF_j) = [deptimeF_j^0, \infty]$;
$VarCancelF_j = 1$, if flight j is cancelled, 0 o.w.. $Dom(VarCancelF_j) = \{0, 1\}$;
$VarSuccAF_j$ – Successor flight after j with the same aircraft. $Dom(VarSuccAF_j) = F$;
$VarPredAF_j$ – Predecessor flight after j with the same aircraft. $Dom(VarPredAF_j) = F$;
$VarArrTF_j$ – arrival time slot of flight j. $Dom(VarArrTF_j) = T_p$ (p: destination airport of flight j);
$VarDepTF_j$ – departure time slot of flight j. $Dom(VarDepTF_j) = T_p$ (p: origin airport of flight j);

Constraints:

#1: $VarArrTimeF_j = VarDepTimeF_j + d_j$;
#2: $VarAF_j = 0 \leftrightarrow VarCancelF_j = 1$;
#3: $VarAF_j > 0 \leftrightarrow VarCancelF_j = 0$;
#4: $VarSuccAF_j = k \leftrightarrow VarPredAF_k = j$;
#5: AllDiff ($VarSuccAF_j$);
#6: AllDiff ($VarPredAF_j$);
#7: $VarSuccAF_j = k \rightarrow (VarDepTimeF_k \geq VarArrTimeF_j + tr[VarAF_j])$ && $VarCancelF_j = 0$;
#8: $VarSuccAF_j = k \rightarrow VarAF_j = VarAF_k$;
#9: $VarAF_j \neq VarAF_k \rightarrow VarSuccAF_j \neq k$;
#10: destination $F_j \neq$ origin $F_j \rightarrow VarSuccAF_j \neq k$;
#11: j and k is an obligatory multileg flight $\rightarrow VarSuccAF_j = k$
#12: j is an virtual maintenance flight $\rightarrow VarAF_j = i$ (i: original aircraft);
#13: j is an virtual breakdown flight $\rightarrow VarAF_j = i$ (i: original aircraft);
#14: $VarArrTimeF_j = arrTimeF_j^0 + delay_j$;
#15: $VarDepTimeF_j = depTimeF_j^0 + delay_j$;
#16: $VarDepTimeF_j \geq depTimeF_j^0$;
#17: $depTimeF_j \leq$ begin of RW $\rightarrow VarDepTimeF_j = depTimeF_j^0$;
#18: $arrTimeF_j \leq$ begin of RW $\rightarrow VarArrTimeF_j = arrTimeF_j^0$;
#19: $VarCancelF_j = 1 \rightarrow VarDepTF_j = -1$;
#20: $VarCancelF_j = 1 \rightarrow VarArrTF_j = -1$;
#21: $VarDepTimeF_j \geq lowerBoundTime_{pt}$ && $VarDepTimeF_j < upperBoundTime_{pt}$ && $VarCancelF_j = 0 \rightarrow VarDepTF_j = t$;
#22: $VarArrTimeF_j \geq lowerBoundTime_{pt}$ && $VarArrTimeF_j < upperBoundTime_{pt}$ && $VarCancelF_j = 0 \rightarrow VarArrTF_j = t$;
#23: Count ($VarDepTF_j = t$) $\leq capAirportDep_{pt}$ (departures from airport p);
#24: Count ($VarArrTF_j = t$) $\leq capAirportArr_{pt}$ (departures from airport p).

O código fonte de todo o trabalho, desde a leitura de dados as Instances do ROADEF, suas devidas conversões, links com a linguagem do ILOG, até a programação do algoritmo em si, pode ser vista no ANEXO.

3.4 *Resultados*

Poucos resultados concretos foram obtidos, devido a problemas de memória, de sintaxe de programação e do fato de ter programado somente uma parte do problema todo, sendo assim, impossível gerar uma resposta que pudesse ser checada pelos "Checkers" fornecidos pelo ROADEF ou sequer comparada com as já obtidas pelo sotware programado em linguagem imperativa.

É esperado, segundo minha co-orientadora Catherine Mancel, que alguém finalize o trabalho e, então, possa se comparar resultados concretos. O que se têm, por enquanto, são apenas resultados parciais que indicam uma melhora no geral.

4 DIFICULDADES ENCONTRADAS

- Manter o alto custo de vida europeu sem bolsa de estudos
- Inglês técnico
- Aprender uma nova linguagem de programação
- Inconsistência no padrão de dados fornecidos pelo *ROADEF 2009 Challenge*
- Vários problemas com a sintaxe da linguagem de programação
- Falta de memória suficiente para realizar testes com *Instances* mais complexas

5 COMENTÁRIOS E CONCLUSÕES

Muito embora tenha passado grande parte do meu tempo de estágio programando, e resolvendo problemas inerentes à programação pura, creio que foi assaz proveitoso o contato com o conhecimento dos pesquisadores com os quais trabalhei, acerca de transporte aéreo e principalmente acerca da otimização, análise e/ou criação de soluções para os mais diversos problemas que afetam o tráfego aéreo.

Infelizmente, como o ITA não possui o software IBM ILOG Solver, não foi possível a aplicação do software para qualquer caso brasileiro, até porque teria que haver um base de dados consistente com a fornecida pelo ROADEF (pois o programa realiza esse tipo de leitura). No entanto, é possível utilizar o conhecimento teórico adquirido para a análise crítica de casos e/ou rotas nacionais.

Além disso, o contato com outros centros de excelência em engenharia (não só a ENAC, mas também a SUPAERO, que fica bem perto e onde há vários alunos do ITA fazendo estágios), assim como seus alunos, mostrou-se bastante interessante para o meu engrandecimento profissional e pessoal.

6 ANEXO

Abaixo, caso seja de interesse, encontra-se todo o código fonte o qual eu desenvolvi durante o meu estágio.

```
//Includes
#include <ilsolver/ilopsolver.h>
#include <ilsolver/ilosolverany.h>
#include <ilsolver/ilosolverfloat.h>
#include <ilsolver/ilosolverint.h>
#include <ilsolver/ilosolverset.h>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <fstream>
#include <iterator>
#include <math.h>
#include <sstream>
#include <stdexcept>
#include <string>
#include <time.h>
#include <vector>

using namespace std;
ILOSTLBEGIN

//Global Definitions
typedef vector<string>          LINE;
typedef vector<int>             LINEINT;
typedef vector<double>          LINEDOUBLE;
typedef vector<LINE>            MATRIX;
typedef vector<LINEINT>         MATRIXI;
typedef vector<LINEDOUBLE>      MATRIXD;

#define MINBYYEAR               365*24*60
#define MINBYMONTH              30*24*60
#define MINBYDAY                24*60
#define MINBYHOUR               60
#define SS                      cin.get();

//FUNCTIONS

//Function to read a "*.csv" file
MATRIX ReadIt(char* arg, string file) //Receive the path, in command line, and the file name as argument
{
        string line; //Stores each line
        string ibs = " "; char tempchar; // string with a blank space and a temporary 'char', to correct some lines of the
original files
        string argv = arg;
        string prefix; //Prefix of the directory where the file must be read
        /* BEGIN - Switch to choose de correct prefix for each instance */
        if (argv == "A01")
                        prefix = "Instances/A01_6088570/";
        else
        if (argv == "A02")
                        prefix = "Instances/A02_6088570/";
        else
        if (argv == "A03")
                        prefix = "Instances/A03_6088570/";
        else
```

```
if (argv == "A04")
                    prefix = "Instances/A04_6088570/";
else
if (argv == "A05")
                    prefix = "Instances/A05_6088570/";
else
if (argv == "A06")
                    prefix = "Instances/A06_6088590/";
else
if (argv == "A07")
                    prefix = "Instances/A07_6088590/";
else
if (argv == "A08")
                    prefix = "Instances/A08_6088590/";
else
if (argv == "A09")
                    prefix = "Instances/A09_6088590/";
else
if (argv == "A10")
                    prefix = "Instances/A10_6088590/";
else
if (argv == "B01")
                    prefix = "Instances/SetB1/B_01/";
else
if (argv == "B02")
                    prefix = "Instances/SetB1/B_02/";
else
if (argv == "B03")
                    prefix = "Instances/SetB1/B_03/";
else
if (argv == "B04")
                    prefix = "Instances/SetB1/B_04/";
else
if (argv == "B05")
                    prefix = "Instances/SetB1/B_05/";
else
if (argv == "B06")
                    prefix = "Instances/SetB2/B_06/";
else
if (argv == "B07")
                    prefix = "Instances/SetB2/B_07/";
else
if (argv == "B08")
                    prefix = "Instances/SetB2/B_08/";
else
if (argv == "B09")
                    prefix = "Instances/SetB2/B_09/";
else
if (argv == "B10")
                    prefix = "Instances/SetB2/B_10/";
else
if (argv == "X01")
                    prefix = "Instances/Instance_X/X01_DATA/";
else
if (argv == "X02")
                    prefix = "Instances/Instance_X/X02_DATA/";
else
if (argv == "X03")
                    prefix = "Instances/Instance_X/X03_DATA/";
else
if (argv == "X04")
                    prefix = "Instances/Instance_X/X04_DATA/";
else
if (argv == "XA01")
                    prefix = "Instances/Instance_X/XA01_DATA/";
else
```

```
	if (argv == "XA02")
				prefix = "Instances/Instance_X/XA02_DATA/";
	else
	if (argv == "XA03")
				prefix = "Instances/Instance_X/XA03_DATA/";
	else
	if (argv == "XA04")
				prefix = "Instances/Instance_X/XA04_DATA/";
	else
	if (argv == "XB01")
				prefix = "Instances/Instance_X/XB01_DATA/";
	else
	if (argv == "XB02")
				prefix = "Instances/Instance_X/XB02_DATA/";
	else
	if (argv == "XB03")
				prefix = "Instances/Instance_X/XB03_DATA/";
	else
	if (argv == "XB04")
				prefix = "Instances/Instance_X/XB04_DATA/";
	else
				prefix = "!!! NOT VALID INSTANCE !!!";
	/* END - Switch to choose de correct prefix for each instance */
	prefix += file; //Merges the directory to to file name. Here we have the full file path to the reading
	int pos = 0; //Position marker
	MATRIX anyth; //Matrix that stores what is going to be read and also the return of the function
	ifstream in(prefix.c_str()); //Opens the file
	if(!in.is_open()) //Check if the file path is valid
	{
		cout << "Failed to open the file: " << prefix << endl; //Warns if the file was not found

	/* Just to stop and wait ENTER to be pressed */
	cout << endl << "Press ENTER to continue..." << endl;
	cin.get();
		/* Just to stop and wait ENTER to be pressed */

    return anyth; //Returns the matrix just to break this function
	}
	while( getline(in,line) ) //While there's no empty line
	{
		/* BEGIN - Insertion of a blank space at the end of the line, if there's not already one, just to read it
correctly after */
		if (line[line.length()-2] != ' ')
		{
			line += ibs;
			tempchar = line[line.length()-2];
			line[line.length()-2] = line[line.length()-1];
			line[line.length()-1] = tempchar;
		}
		/* END - Insertion of a blank space at the end of the line, if there's not already one, just to read it correctly
after */
		LINE new_line; //Creates a new line, that's going to be added to the matrix
		string field; //Temporary field to the itens
		while( (pos = line.find(' ')) >= 0) //While the marker finds a space
		{
			field = line.substr(0,pos); //Adds what is between the beginning of the line and the position of
the marker
			line = line.substr(pos+1); //Updates de line with the content between the position just after the
marker (to cut off the space) and the end of the line
			new_line.push_back(field); //Adds the temporary field as the next in the line
		}
		if (line.length() > 0)
		{
			new_line.push_back(line);
		}
		anyth.push_back(new_line); //Adds the line as the next in the matrix
```

```
        }
        return anyth; //Returns the matrix with the content of the file
}

//Function to print a whole matrix on the screen
void PrintIt (MATRIX anyth) //Receives the matrix as argument
{
        MATRIX::size_type n_lines; //Counter of lines
        LINE::size_type n_columns; //Counter of columns
  for (n_lines = 0; n_lines < anyth.size(); n_lines++) //"n_lines+1" ignores the '#' that ends each file provided by ROADEF
  {
    cout << n_lines << " - "; //NUmbers the lines, starting from zero, evidently
    for (n_columns = 0; n_columns < anyth[n_lines].size(); n_columns++) //Iteration to pass through all columns of the line
    {
      cout << anyth[n_lines][n_columns] << "  \t"; //Prints each element and an empty space as a spacer
    }
    cout << endl << endl; //Skips one line between two matrix's lines
  }
  cout << endl << endl << "Lines: " << anyth.size() << endl << endl; //Prints the number of lines
        cout << "Columns: " << anyth[0].size() << endl << endl << endl << endl;//And columns, just to check with the
originals
}

//Function to print a whole matrix of doubles on the screen
void PrintItD (MATRIXD anyth) //Receives the matrixd as argument
{
        int n_lines; //Counter of lines
        int n_columns; //Counter of columns
  for (n_lines = 0; n_lines < anyth.size(); n_lines++) //"n_lines+1" ignores the '#' that ends each file provided by ROADEF
  {
    cout << n_lines << " - "; //Numbers the lines, starting from zero, evidently
    for (n_columns = 0; n_columns < anyth[n_lines].size(); n_columns++) //Iteration to pass through all columns of the line
    {
      cout << anyth[n_lines][n_columns] << "  "; //Prints each element and an empty space as a spacer
    }
    cout << endl << endl; //Skips one line between two matrix's lines
  }
  cout << endl << endl << "Linhas: " << anyth.size() << endl << endl; //Prints the number of lines
  cout << "Colunas: " << anyth[0].size() << endl << endl << endl;//And columns, just to check with the originals
}

//Function to create a new "*.csv" file from a given matrix
void CreateCSV (MATRIX anyth, string file_name) //Receives as arguments: the matrix to be converted into a file and the
name of the original file
{
  string prefix = "Instances/A01_6088570/@New_"; //Prefix of the directory where the file must be created
  file_name += ".csv"; //Adds de wanted extension, in this case: '.csv'
  prefix += file_name; //Merges the directory to to file name. Here we have the full file path to the creation
  ofstream out(prefix.c_str()); //Creates the file
  MATRIX::size_type n_lines = 0; //Counter of lines
  LINE::size_type n_columns = 0; //Counter of columns
  for (n_lines = 0; n_lines+1 < anyth.size(); n_lines++) //"n_lines+1" ignores the '#' that ends each file provided by
ROADEF
  {
    for (n_columns = 0; n_columns < anyth[n_lines].size(); n_columns++) //Iteration to pass through all columns of the line
    {
      out << anyth[n_lines][n_columns] << " "; //Prints each element and an empty space as a spacer
    }
    out << endl; //Starts a new line
  }
  out << "#"; //Prints the '#' on the last line, as in the original files
  out.close(); //Closes the file
}

//Function to convert a LINE into a LINEINT (to be able to do arithmetical operations)
```

```
LINEINT ConvToInt (MATRIX anyth, int column) //Receives the matrix and the number of it's column that must be converted
{
    int i, temp; //iterator and temporary field
    LINEINT out_line; //Vector of integers that is going to be the return of the function
    if (column >= anyth[0].size()) //Warns if the column number passed doesn't exist in the matrix
    {
        cout << "Column doesn't exist!" << endl; //Warning
        /* Just to stop and wait ENTER to be pressed */
        cout << endl << "Press ENTER to continue..." << endl;
        cin.get();
        /* Just to stop and wait ENTER to be pressed */
    }
    for (i = 0; i+1 < anyth.size(); i++) //"i+1" ignores the '#' that ends each file provided by ROADEF
    {
      istringstream iss(anyth[i][column]); //Extracts the field as an integer
      iss >> temp; //Copies it to the temporary field
      out_line.push_back(temp); //Adds the integer as the next item in the vector
    }
    return out_line; //Returns the vector of ints
}

//Function to convert a LINEINT into a LINE and update the matrix
MATRIX PutInMat (MATRIX anyth, LINEINT conv, int column) //Receives as arguments: the matrix to be updated, the vector of ints and the column number in the matrix
{
    int i, j, temp; //Iterators and temporary field
    LINE new_line; //Vector of strings that is going to be used to update de matrix
    for (i = 0; i+1 < anyth.size(); i++) //"i+1" ignores the '#' that ends each file provided by ROADEF
    {
      stringstream buff; // Creates a buffer to extract the integer as a string
      buff << conv[i]; // Extracts each element
      new_line.push_back(buff.str()); //Add each element as the next string item in the vector
    }
    for (j = 0; j+1 < anyth.size(); j++) //"j+1" ignores the '#' that ends each file provided by ROADEF
    {
      anyth[j][column] = new_line[j]; //Updates the wanted column in the matrix
    }
    return anyth; //Returns the updated matrix
}

//Function to get the recovery period in the "config.csv"
LINEINT RecPeriod (MATRIX anyth) //Receives the matrix "config" as argument
{
        LINE temp_line; //Temporary vector of strings
        LINEINT out_line; //Vector of integers that will be returned
        int i, pos = 0, temp_int; //Iterator, position marker and temporary field
        /* BEGIN - Reading and split of the values of date and time */
        for (i = 0; i < 4; i++)
        {
                while( (pos = anyth[0][i].find_first_of("///:")) != anyth[0][i].npos )
                {
                        if (pos > 0)
                        {
                                temp_line.push_back(anyth[0][i].substr(0,pos));
                        }
                        anyth[0][i] = anyth[0][i].substr(pos+1);
                }
                if (anyth[0][i].length() > 0)
                {
                        temp_line.push_back(anyth[0][i]);
                }
        }
        /* END - Reading and split of the values of date and time */
        /* BEGIN - Conversion to integers */
        for (i = 0; i < temp_line.size(); i++)
```

```
        {
    istringstream iss(temp_line[i]); //Extracts the field as an integer
    iss >> temp_int; //Copies it to the temporary field
    out_line.push_back(temp_int); //Adds the integer as the next item in the vector
        }
        /* END - Conversion to integers */
        return out_line; //Return of the vector of integers
}

// Function to get the costs of delays in the "config.csv"
LINEDOUBLE ConfigDelay (MATRIX anyth) //Receives the matrix "config" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
        double temp; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 2; i < anyth[1].size(); i += 3)
        {
                temp = atof (anyth[1][i].c_str());
                out_line.push_back (temp);
        }
        /* END - conversion to doubles */
        return out_line; //Return of the vector of doubles
}

// Function to get the costs of cancellations on outbound trips in the "config.csv"
LINEDOUBLE ConfigOutbound (MATRIX anyth) //Receives the matrix "config" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
        double temp; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 2; i < anyth[2].size(); i += 3)
        {
                temp = atof (anyth[2][i].c_str());
                out_line.push_back (temp);
        }
        /* END - conversion to doubles */
        return out_line; //Return of the vector of doubles
}

// Function to get the costs of cancellations on inbound trips or connections in the "config.csv"
LINEDOUBLE ConfigInbound (MATRIX anyth) //Receives the matrix "config" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
        double temp; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 2; i < anyth[3].size(); i += 3)
        {
                temp = atof (anyth[3][i].c_str());
                out_line.push_back (temp);
        }
        /* END - conversion to doubles */
        return out_line; //Return of the vector of doubles
}

// Function to get the costs of downgrading in the "config.csv"
LINEDOUBLE ConfigDowngr (MATRIX anyth) //Receives the matrix "config" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
        double temp; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 3; i < anyth[4].size(); i += 4)
        {
```

```
                temp = atof (anyth[4][i].c_str());
                out_line.push_back (temp);
        }
        /* END - conversion to doubles */
        return out_line; //Return of the vector of doubles
}

// Function to get the costs of non-compliant location of an aircraft in the "config.csv"
LINEDOUBLE ConfigNonCompl (MATRIX anyth) //Receives the matrix "config" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
        double temp; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 0; i < anyth[5].size(); i++)
        {
                temp = atof (anyth[5][i].c_str());
                out_line.push_back (temp);
        }
        /* END - conversion to doubles */
        return out_line; //Return of the vector of doubles
}

// Function to get the weights alfa, beta and gama in the "config.csv"
LINEDOUBLE ConfigWeights (MATRIX anyth) //Receives the matrix "config" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
        double temp; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 0; i < anyth[6].size(); i++)
        {
                temp = atof (anyth[6][i].c_str());
                out_line.push_back (temp);
        }
        /* END - conversion to doubles */
        return out_line; //Return of the vector of doubles
}

// Function to get the values as doubles in the "airports.csv"
MATRIXD AirportsValues (MATRIX anyth) //Receives the matrix "airports" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, j, k, pos = 0; //Iterators and position counters
        double temp_dble; //Temporary field
        /* BEGIN - Extraction of each number in one field */
        for (i = 0; i+1 < anyth.size(); i++)
        {
                LINE temp_line; //Temporary vector of strings
                for (j = 1; j < anyth[i].size(); j++)
                {
                        while( (pos = anyth[i][j].find_first_of("/:")) != anyth[0][i].npos )
                        {
                                if (pos > 0)
                                {
                                        temp_line.push_back(anyth[i][j].substr(0,pos));
                                }
                                anyth[i][j] = anyth[i][j].substr(pos+1);
                        }
                        if (anyth[i][j].length() > 0)
                        {
                                temp_line.push_back(anyth[i][j]);
                        }
                }
                /* BEGIN - Conversion to double */
                LINEDOUBLE dble_line; //Temporary vector of doubles
```

```
                for (k = 0; k < temp_line.size(); k++)
                {
                istringstream iss(temp_line[k]);
                iss >> temp_dble;
                dble_line.push_back(temp_dble);
                }
                /* END - Conversion to double */
                out_mat.push_back(dble_line);
        }
        /* END - Extraction of each number in one field */
        return out_mat; //Return of the matrix of doubles
}

// Function to get the departure capacities per time slot in the "airports.csv"
MATRIXD AirportsDepCap (MATRIXD anyth) //Receives the matrix "airports_values" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, j, k, nts; //Iterator
        for (i = 0; i < anyth.size(); i++)
        {
                LINEDOUBLE dble_line; //Temporary vector of doubles
                for (j = 4; j < anyth[i].size(); j += 6)
                {
                        nts = anyth[i][j] - anyth[i][j-2];
                        if (nts < 0)
                        {
                                nts += 24;
                        }
                        for (k = 0; k < nts; k++)
                        {
                                dble_line.push_back(anyth[i][j-4]);
                        }
                }
                out_mat.push_back(dble_line);
        }
        return out_mat; //Return of the matrix of doubles
}

// Function to get the arrival capacities per time slot in the "airports.csv"
MATRIXD AirportsArrCap (MATRIXD anyth) //Receives the matrix "airports_values" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, j, k, nts; //Iterator
        for (i = 0; i < anyth.size(); i++)
        {
                LINEDOUBLE dble_line; //Temporary vector of doubles
                for (j = 4; j < anyth[i].size(); j += 6)
                {
                        nts = anyth[i][j] - anyth[i][j-2];
                        if (nts < 0)
                        {
                                nts += 24;
                        }
                        for (k = 0; k < nts; k++)
                        {
                                dble_line.push_back(anyth[i][j-3]);
                        }
                }
                out_mat.push_back(dble_line);
        }
        return out_mat; //Return of the matrix of doubles
}

// Function to get the flight time in the "dist.csv"
LINEDOUBLE DistTime (MATRIX anyth) //Receives the matrix "dist" as argument
{
```

```
	LINEDOUBLE out_line; //Vector of doubles that will be returned
	int i; //Iterator
	double temp; //Temporary field
	/* BEGIN - conversion to doubles */
	for (i = 0; i < anyth[2].size(); i++)
	{
		temp = atof (anyth[2][i].c_str());
		out_line.push_back (temp);
	}
	/* END - conversion to doubles */
	return out_line; //Return of the vector of doubles
}

// Function to get the arrival and departure times in the "flights.csv"
MATRIXD FlightsTimes (MATRIX anyth) //Receives the matrix "flights" as argument
{
	MATRIXD out_mat; //Matrix of doubles that will be returned
	int i, j, k, pos = 0, pos2 = 0; //Iterators and position counters
	double temp_dble; //Temporary field
	/* BEGIN - Extraction of each number in one field */
	for (i = 0; i+1 < anyth.size(); i++)
	{
		LINE temp_line; //Temporary vector of strings
		for (j = 3; j < anyth[i].size(); j++)
		{
			while( (pos = anyth[i][j].find_first_of("/:")) != anyth[0][i].npos )
			{
				if (pos > 0)
				{
					temp_line.push_back(anyth[i][j].substr(0,pos));
				}
				anyth[i][j] = anyth[i][j].substr(pos+1);
			}
			if (anyth[i][j].length() > 0)
			{
				temp_line.push_back(anyth[i][j]);
			}
		}
		temp_line.push_back(temp_line[4]);
		/* BEGIN - Creation of one column to specify the changing of day */
		if (temp_line[3].length() > 2) //If the field has more than 2 characters, it means that it has a '+1', so...
		{
			while( (pos2 = temp_line[3].find_first_of("/+")) != temp_line[3].npos ) // Finds the '+' signal
			{
				if (pos2 > 0)
				{
					temp_line[3] = temp_line[3].substr(0,pos2); //Updates de field just with the
time
				}
			}
			temp_line[4] = "1"; //Write '1' in the next field to show that the flight arrives on the following
day
		}
		else //In the other hand, if the field has no more than 2 characters...
		{
			temp_line[4] = "0"; //Write '0' in the next field to show that the flight arrives on the same day
		}
		/* END - Creation of one column to specify the changing of day */
		/* BEGIN - Conversion to double */
		LINEDOUBLE dble_line; //Temporary vector of doubles
		for (k = 0; k < temp_line.size(); k++)
		{
		istringstream iss(temp_line[k]);
		iss >> temp_dble;
		dble_line.push_back(temp_dble);
		}
```

```
                /* END - Conversion to double */
                out_mat.push_back(dble_line);
        }
        /* END - Extraction of each number in one field */
        return out_mat; //Return of the matrix of doubles
}

//Function to get the configuration in the "aircraft.csv"
MATRIXD AircraftConfig (MATRIX anyth) //Receives the matrix "aircraft" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, k, pos = 0; //Iterators and position counter
        double temp_dble; //Temporary field
        /* BEGIN - Split the cabin configuration into 3 fields */
        for (i = 0; i+1 < anyth.size(); i++)
        {
                LINE temp_line; //Temporary vector of strings
                while( (pos = anyth[i][3].find_first_of("//")) != anyth[i][3].npos )
                {
                        if (pos > 0)
                        {
                                temp_line.push_back(anyth[i][3].substr(0,pos));
                        }
                        anyth[i][3] = anyth[i][3].substr(pos+1);
                }
                if (anyth[i][3].length() > 0)
                {
                        temp_line.push_back(anyth[i][3]);
                }
                /* BEGIN - Conversion to double */
                LINEDOUBLE dble_line; //Temporary vector of doubles
                for (k = 0; k < temp_line.size(); k++)
                {
                istringstream iss(temp_line[k]);
                iss >> temp_dble;
                dble_line.push_back(temp_dble);
                }
                /* END - Conversion to double */
                out_mat.push_back(dble_line);

        }
        /* END - Split the cabin configuration into 3 fields */
        return out_mat; //Return of the matrix of doubles
}

// Function to get the range in flight hours in the "dist.csv"
LINEDOUBLE AircraftRange (MATRIX anyth) //Receives the matrix "dist" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
        double temp; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 0; i+1 < anyth.size(); i++)
        {
                temp = atof (anyth[i][4].c_str());
                out_line.push_back (temp);
        }
        /* END - conversion to doubles */
        return out_line; //Return of the vector of doubles
}

// Function to get the hourly operating costs in the "dist.csv"
LINEDOUBLE AircraftHCost (MATRIX anyth) //Receives the matrix "dist" as argument
{
        LINEDOUBLE out_line; //Vector of doubles that will be returned
        int i; //Iterator
```

```
	double temp; //Temporary field
	/* BEGIN - conversion to doubles */
	for (i = 0; i+1 < anyth.size(); i++)
	{
		temp = atof (anyth[i][5].c_str());
		out_line.push_back (temp);
	}
	/* END - conversion to doubles */
	return out_line; //Return of the vector of doubles
}

// Function to get the turnround time in the "dist.csv"
LINEDOUBLE AircraftTurnRound (MATRIX anyth) //Receives the matrix "dist" as argument
{
	LINEDOUBLE out_line; //Vector of doubles that will be returned
	int i; //Iterator
	double temp; //Temporary field
	/* BEGIN - conversion to doubles */
	for (i = 0; i+1 < anyth.size(); i++)
	{
		temp = atof (anyth[i][6].c_str());
		out_line.push_back (temp);
	}
	/* END - conversion to doubles */
	return out_line; //Return of the vector of doubles
}

// Function to get the transit time in the "dist.csv"
LINEDOUBLE AircraftTransit (MATRIX anyth) //Receives the matrix "dist" as argument
{
	LINEDOUBLE out_line; //Vector of doubles that will be returned
	int i; //Iterator
	double temp; //Temporary field
	/* BEGIN - conversion to doubles */
	for (i = 0; i+1 < anyth.size(); i++)
	{
		temp = atof (anyth[i][7].c_str());
		out_line.push_back (temp);
	}
	/* END - conversion to doubles */
	return out_line; //Return of the vector of doubles
}

//Function to get the maintenance in the "aircraft.csv"
MATRIX AircraftMaint (MATRIX anyth) //Receives the matrix "aircraft" as argument
{
	MATRIX out_mat; //Matrix of strings that will be returned
	int i, pos = 0; //Iterator and position counter
	/* BEGIN - Split of all fields on new fields */
	for (i = 0; i+1 < anyth.size(); i++)
	{
		LINE temp_line; //Temporary vector of strings
		if (anyth[i][9] == "NULL" || anyth[i][9] == "NUL") //Fullfil the blank spaces with "NULL" if there's no
maintenance **"NUL" is a problem in some instances that they provided (X01, for exemple)
		{
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
			temp_line.push_back("NULL");
```

```
                        temp_line.push_back("NULL");
                }
                else //Otherwise, if there's a maintenance predicted
                {
                        while( (pos = anyth[i][9].find_first_of("///-/:")) != anyth[i][9].npos )
                        {
                                if (pos > 0)
                                {
                                        temp_line.push_back(anyth[i][9].substr(0,pos));
                                }
                                anyth[i][9] = anyth[i][9].substr(pos+1);
                        }
                        if (anyth[i][9].length() > 0)
                        {
                                temp_line.push_back(anyth[i][9]);
                        }
                }
                out_mat.push_back(temp_line);

        }
        /* END - Split of all fields on new fields */
        return out_mat; //Return of the matrix of strings
}

//Function to get the maintenance values in the "aircraft.csv"
MATRIXD AircraftMaintenance (MATRIX anyth) //Receives the matrix "aircraft_maint" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, k, pos = 0; //Iterators and position counter
        double temp_dble; //Temporary field
        /* BEGIN - conversion to doubles */
        for (i = 0; i < anyth.size(); i++)
        {
                LINEDOUBLE dble_line; //Temporary vector of doubles
                for (k = 1; k < anyth[i].size(); k++)
                {
                        if (anyth[i][k] == "NULL")
                        {
                                dble_line.push_back(0); //If there's no maintenance, all fields receives the value '0'

                        }
                        else //Otherwise, the fields receives the values
                        {
                                istringstream iss(anyth[i][k]);
                                iss >> temp_dble;
                                dble_line.push_back(temp_dble);
                        }
                }
                out_mat.push_back(dble_line);
        }
        /* END - conversion to doubles */
        return out_mat; //Return of the matrix of doubles
}

//Function to get the date of rotations in the "rotations.csv"
MATRIXD RotationsDate (MATRIX anyth) //Receives the matrix "rotations" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, j, pos = 0; //Iterators and position counter
        double temp;        //Temporary field
        for (i = 0; i+1 < anyth.size(); i++)
        {
                LINE temp_line; //Temporary vector of strings
                while( (pos = anyth[i][1].find_first_of("//")) != anyth[i][1].npos )
                {
                        if (pos > 0)
```

```
                {
                        temp_line.push_back(anyth[i][1].substr(0,pos));
                }
                anyth[i][1] = anyth[i][1].substr(pos+1);
        }
        if (anyth[i][1].length() > 0)
        {
                temp_line.push_back(anyth[i][1]);
        }
        /* BEGIN - conversion to doubles */
        LINEDOUBLE dble_line; //Temporary vector of doubles
        for (j = 0; j < temp_line.size(); j++)
        {
            istringstream iss(temp_line[j]);
        iss >> temp;
                dble_line.push_back(temp);
        }
        /* END - conversion to doubles */
        out_mat.push_back(dble_line);
    }
    return out_mat; //Return of the matrix of doubles
}

// Function to get the itinerary identification number in the "itineraries.csv"
LINEINT ItineraryIDN (MATRIX anyth) //Receives the matrix "itineraries" as argument
{
    LINEINT out_line; //Vector of integers that will be returned
    int i, temp; //Iterator and temporary field
    /* BEGIN - conversion to integers */
    for (i = 0; i+1 < anyth.size(); i++)
    {
        temp = atoi (anyth[i][0].c_str());
        out_line.push_back (temp);
    }
    /* END - conversion to integers */
    return out_line; //Return of the vector of integers
}

// Function to get the unit price in the "itineraries.csv".
LINEDOUBLE ItineraryPrice (MATRIX anyth) //Receives the matrix "itineraries" as argument
{
    LINEDOUBLE out_line; //Vector of doubles that will be returned
    int i; //Iterator
    double temp; //Temporary field
    /* BEGIN - conversion to doubles */
    for (i = 0; i+1 < anyth.size(); i++)
    {
        temp = atof (anyth[i][2].c_str());
        out_line.push_back (temp);
    }
    /* END - conversion to doubles */
    return out_line; //Return of the vector of doubles
}

// Function to get the number of passengers in the "itineraries.csv"
LINEINT ItineraryPassengers (MATRIX anyth) //Receives the matrix "itineraries" as argument
{
    LINEINT out_line; //Vector of integers that will be returned
    int i, temp; //Iterator and temporary field
    /* BEGIN - conversion to integers */
    for (i = 0; i+1 < anyth.size(); i++)
    {
        temp = atoi (anyth[i][3].c_str());
        out_line.push_back (temp);
    }
    /* END - conversion to integers */
```

```
	return out_line; //Return of the vector of integers
}

//Function to get the flights, dates and cabins in the "itineraries.csv"
MATRIXD ItineraryFlightsDates (MATRIX anyth) //Receives the matrix "itineraries" as argument
{
	MATRIXD out_mat; //Matrix of doubles that will be returned
	int i, j, k, pos = 0; //Iterators and position counter
	double temp; //Temporary field
	for (i = 0; i+1 < anyth.size(); i++)
	{
		LINE temp_line; //Temporary vector of strings
		for (j = 4; j+1 < anyth[i].size(); j++)
		{
			/* BEGIN - Conversion of the cabins letters to numbers */
			if (anyth[i][j] == "F")
			{
				anyth[i][j] = "1";
			}
			if (anyth[i][j] == "B")
			{
				anyth[i][j] = "2";
			}
			if (anyth[i][j] == "E")
			{
				anyth[i][j] = "3";
			}
			/* END - Conversion of the cabins letters to numbers */
			/* BEGIN - Split of each field in new ones, if there's more than one value */
			while( (pos = anyth[i][j].find_first_of("//")) != anyth[i][j].npos )
			{
				if (pos > 0)
				{
					temp_line.push_back(anyth[i][j].substr(0,pos));
				}
				anyth[i][j] = anyth[i][j].substr(pos+1);
			}
			if (anyth[i][j].length() > 0)
			{
				temp_line.push_back(anyth[i][j]);
			}
			/* END - Split of each field in new ones, if there's more than one value */
		/* BEGIN - conversion to doubles */
		LINEDOUBLE dble_line; //Temporary vector of doubles
		for (k = 0; k < temp_line.size(); k++)
		{
		    istringstream iss(temp_line[k]);
		iss >> temp;
			dble_line.push_back(temp);
		}
		/* END - conversion to doubles */
		out_mat.push_back(dble_line);
		}
	}
	return out_mat; //Return of the matrix of doubles
}

//Function to rearrange the matrix "position" in a better format
MATRIX PositionRearrange (MATRIX anyth) //Receives the matrix "position" as argument
{
	MATRIX out_mat; //Matrix of strings that will be returned
	int i, j, k, pos = 0; //Iterators and position counter
	for (i = 0; i+1 < anyth.size(); i++)
	{
		/* BEGIN - Creation of a new line to each aircraft required */
		int x_times = (anyth[i].size()-2)/3; //Number of new lines to be created
```

```
                for (j = 0; j < x_times; j++)
                {
                        LINE temp_line; //Temporary vector of strings
                        temp_line.push_back(anyth[i][0]); //Putting of the airport 3 letters in the first field of each line
                        for (k = 1; k < 4; k++)
                        {
                                temp_line.push_back(anyth[i][3*j + k]);
                        }
                        out_mat.push_back(temp_line);
                }
                /* END - Creation of a new line to each aircraft required */
        }
        return out_mat; //Return of the matrix of strings
}

//Function to get cabin configuration and the number of aircrafts in the "position.csv" (actually, from the matrix 'position2')
MATRIXD Position2Cabin (MATRIX anyth) //Receives the matrix "position2" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, j, k, pos = 0; //Iterators and position counter
        double temp; //Temporary field
        for (i = 0; i < anyth.size(); i++)
        {
                LINE temp_line; //Temporary vector of strings
                /* BEGIN - Split of the fields with more than one value into new ones */
                for (j = 2; j < anyth[i].size(); j++)
                {
                        while( (pos = anyth[i][j].find_first_of("//")) != anyth[i][j].npos )
                        {
                                if (pos > 0)
                                {
                                        temp_line.push_back(anyth[i][j].substr(0,pos));
                                }
                                anyth[i][j] = anyth[i][j].substr(pos+1);
                        }
                        if (anyth[i][j].length() > 0)
                        {
                                temp_line.push_back(anyth[i][j]);
                        }
                }
                /* END - Split of the fields with more than one value into new ones */
                /* BEGIN - conversion to doubles */
                LINEDOUBLE dble_line; //Temporary vector of doubles
                for (k = 0; k < temp_line.size(); k++)
                {
                    istringstream iss(temp_line[k]);
                iss >> temp;
                        dble_line.push_back(temp);
                }
                /* END - conversion to doubles */
                out_mat.push_back(dble_line);
        }
        return out_mat; //Return of the matrix of doubles
}

//Function to get all data in the "alt_flights.csv"
MATRIXD AltFlightsAll (MATRIX anyth) //Receives the matrix "alt_flights" as argument
{
        MATRIXD out_mat; //Matrix of doubles that will be returned
        int i, j, k, pos = 0; //Iterators and position counter
        double temp; //Temporary field
        for (i = 0; i+1 < anyth.size(); i++)
        {
                LINE temp_line; //Temporary vector of strings
                /* BEGIN - Split of the fields with more than one value into new ones */
                for (j = 0; j < anyth[i].size(); j++)
```

```
		{
			while( (pos = anyth[i][j].find_first_of("//")) != anyth[i][j].npos )
			{
				if (pos > 0)
				{
					temp_line.push_back(anyth[i][j].substr(0,pos));
				}
				anyth[i][j] = anyth[i][j].substr(pos+1);
			}
			if (anyth[i][j].length() > 0)
			{
				temp_line.push_back(anyth[i][j]);
			}
		}
		/* END - Split of the fields with more than one value into new ones */
		/* BEGIN - conversion to doubles */
		LINEDOUBLE dble_line; //Temporary vector of doubles
		for (k = 0; k < temp_line.size(); k++)
		{
		    istringstream iss(temp_line[k]);
		iss >> temp;
			dble_line.push_back(temp);
		}
		/* END - conversion to doubles */
		out_mat.push_back(dble_line);
	}
	return out_mat; //Return of the matrix of doubles
}

//Function to dates and times in the "alt_aircraft.csv"
MATRIXD AltAircraftDates (MATRIX anyth) //Receives the matrix "alt_aircraft" as argument
{
	MATRIXD out_mat; //Matrix of doubles that will be returned
	int i, j, k, pos = 0; //Iterators and position counter
	double temp; //Temporary field
	for (i = 0; i+1 < anyth.size(); i++)
	{
		LINE temp_line; //Temporary vector of strings
		/* BEGIN - Split of the fields with more than one value into new ones */
		for (j = 1; j < anyth[i].size(); j++)
		{
			while( (pos = anyth[i][j].find_first_of("///:")) != anyth[i][j].npos )
			{
				if (pos > 0)
				{
					temp_line.push_back(anyth[i][j].substr(0,pos));
				}
				anyth[i][j] = anyth[i][j].substr(pos+1);
			}
			if (anyth[i][j].length() > 0)
			{
				temp_line.push_back(anyth[i][j]);
			}
		}
		/* END - Split of the fields with more than one value into new ones */
		/* BEGIN - conversion to doubles */
		LINEDOUBLE dble_line; //Temporary vector of doubles
		for (k = 0; k < temp_line.size(); k++)
		{
		    istringstream iss(temp_line[k]);
		iss >> temp;
			dble_line.push_back(temp);
		}
		/* END - conversion to doubles */
		out_mat.push_back(dble_line);
	}
```

```
	return out_mat; //Return of the matrix of doubles
}

//Function to dates, times and capacities in the "alt_airports.csv"
MATRIXD AltAirportsAll (MATRIX anyth) //Receives the matrix "alt_airports" as argument
{
	MATRIXD out_mat; //Matrix of doubles that will be returned
	int i, j, k, pos = 0; //Iterators and position counter
	double temp; //Temporary field
	for (i = 0; i+1 < anyth.size(); i++)
	{
		LINE temp_line; //Temporary vector of strings
		/* BEGIN - Split of the fields with more than one value into new ones */
		for (j = 1; j+1 < anyth[i].size(); j++)
		{
			while( (pos = anyth[i][j].find_first_of("///:")) != anyth[i][j].npos )
			{
				if (pos > 0)
				{
					temp_line.push_back(anyth[i][j].substr(0,pos));
				}
				anyth[i][j] = anyth[i][j].substr(pos+1);
			}
			if (anyth[i][j].length() > 0)
			{
				temp_line.push_back(anyth[i][j]);
			}
		}
		/* END - Split of the fields with more than one value into new ones */
		/* BEGIN - conversion to doubles */
		LINEDOUBLE dble_line; //Temporary vector of doubles
		for (k = 0; k < temp_line.size(); k++)
		{
		    istringstream iss(temp_line[k]);
		iss >> temp;
			dble_line.push_back(temp);
		}
		/* END - conversion to doubles */
		out_mat.push_back(dble_line);
	}
	return out_mat; //Return of the matrix of doubles
}

//Function to get the index of aircraft
int GetIndexAircraft (MATRIX anyth, string name)
{
	int result = -1;
	for (int j = 0; j+1 < anyth.size(); j++)
	{
		if (anyth[j][0] == name)
		{
			return j+1;
		}
	}
 return result;
}

//Function to get the index of flight
int GetIndexFlight (MATRIX anyth, double x)
{
	string name;
	int result = -1;
	ostringstream oss;
    oss << x;
    name = oss.str();
	for (int j = 0; j+1 < anyth.size(); j++)
```

```
	{
		if (anyth[j][0] == name)
		{
			return j;
		}
	}
 return result;
}

//Function to get the index of flight
int GetIndexFlight2 (MATRIX anyth, string name)
{
	int result = -1;
	for (int j = 0; j+1 < anyth.size(); j++)
	{
		if (anyth[j][0] == name)
		{
			return j;
		}
	}
 return result;
}

//Function to get the index of flight in flights
int GetIndexRotFl (double x, MATRIX flights)
{
	string name;
	int result = -1;
	ostringstream oss;
      oss << x;
      name = oss.str();
	for (int j = 0; j+1 < flights.size(); j++)
	{
		if (flights[j][0] == name)
		{
			return j;
		}
	}
  return result;
}

//Function to get the time, in minutes, between midnight of the day of the first flight and the end of the recovery period
int GetUpperBound (MATRIXD rotations_date, LINEINT anyth_line) //Receives the matrix "rotations_date" and the line
"config_rec_period"
{
	int i, out = 0;
	int year_base = rotations_date[0][2], month_base = rotations_date[0][1], day_base = rotations_date[0][0]; //Date of
the earlier flight in "rotations"
	for (i = 0; i < rotations_date.size(); i++) //For all flights in "rotations"
	{
		/* BEGIN - Look for the earlier day */
		if ( year_base > rotations_date[i][2] )
		{
			day_base = rotations_date[i][0];
			month_base = rotations_date[i][1];
			year_base = rotations_date[i][2];
		}
		if ( year_base == rotations_date[i][2] )
		{
			if ( month_base > rotations_date[i][1] )
			{
				day_base = rotations_date[i][0];
				month_base = rotations_date[i][1];
			}
			if ( month_base == rotations_date[i][1] )
			{
```

```
                                        if ( day_base > rotations_date[i][0] )
                                        {
                                                day_base = rotations_date[i][0];
                                        }
                                }
                        }
                        /* END - Look for the earlier day */
                }
                /* BEGIN - Calculating the difference */
                out += (anyth_line[5] - day_base) * MINBYDAY;
                out += (anyth_line[6] - month_base) * MINBYMONTH;
                out += (anyth_line[7] - year_base) * MINBYYEAR;
                out += anyth_line[8]*MINBYHOUR + anyth_line[9];
                /* END - Calculating the difference */
                return out;
}

//Function to get the time, in minutes, between midnight of the day of the first flight and the beginning of the recovery period
int GetLowerBound (MATRIXD rotations_date, LINEINT anyth_line) //Receives the matrix "rotations_date" and the line "config_rec_period"
{
        int i, out = 0;
        int year_base = rotations_date[0][2], month_base = rotations_date[0][1], day_base = rotations_date[0][0]; //Date of the earlier flight in "rotations"
        for (i = 0; i < rotations_date.size(); i++) //For all flights in "rotations"
        {
                /* BEGIN - Look for the earlier day */
                if ( year_base > rotations_date[i][2] )
                {
                        day_base = rotations_date[i][0];
                        month_base = rotations_date[i][1];
                        year_base = rotations_date[i][2];
                }
                if ( year_base == rotations_date[i][2] )
                {
                        if ( month_base > rotations_date[i][1] )
                        {
                                day_base = rotations_date[i][0];
                                month_base = rotations_date[i][1];
                        }
                        if ( month_base == rotations_date[i][1] )
                        {
                                if ( day_base > rotations_date[i][0] )
                                {
                                        day_base = rotations_date[i][0];
                                }
                        }
                }
                /* END - Look for the earlier day */
        }
        /* BEGIN - Calculating the difference */
        out += (anyth_line[0] - day_base) * MINBYDAY;
        out += (anyth_line[1] - month_base) * MINBYMONTH;
        out += (anyth_line[2] - year_base) * MINBYYEAR;
        out += anyth_line[3]*MINBYHOUR + anyth_line[4];
        /* END - Calculating the difference */
        return out;
}

//Function to create a matrix with Real Flights, Virtual Maintenance and Virtual Breakdown (strings)
MATRIX AllFlightsString (MATRIX rotations, MATRIX flights, MATRIX aircraft, MATRIX aircraft_maint, MATRIX alt_aircraft)
{
        MATRIX out;
        int i, j;
        for (i = 0; i+1 < rotations.size(); i++)
```

```
	{
		LINE temp_line;
		temp_line.push_back ("Real Flight");
		temp_line.push_back (rotations[i][2]);
		for (j = 0; j+1 < flights.size(); j++) //Looking for the flight number in "flights"
		{
			if (rotations[i][0] == flights[j][0]) // When it finds the correct flight
			{
				temp_line.push_back (flights[j][1]);
				temp_line.push_back (flights[j][2]);
			}
		}
		out.push_back (temp_line);
	}
	for (i = 0; i+1 < aircraft.size(); i++)
	{
		if (aircraft_maint[i][0] != "NULL") //Looking for the maintenances
		{
			LINE temp_line;
			temp_line.push_back ("Virtual Maintenance Flight");
			temp_line.push_back (aircraft[i][0]);
			temp_line.push_back (aircraft_maint[i][0]);
			temp_line.push_back (aircraft_maint[i][0]);
			out.push_back (temp_line);
		}
	}
	for (i = 0; i+1 < alt_aircraft.size(); i++) //Looking for the breakdowns
	{
		LINE temp_line;
		temp_line.push_back ("Virtual Breakdown Flight");
		temp_line.push_back (alt_aircraft[i][0]);
		temp_line.push_back ("0");
		temp_line.push_back ("0");
		out.push_back (temp_line);
	}
	return out;
}

//Function to create a matrix with Real Flights, Virtual Maintenance and Virtual Breakdown (values).
MATRIXI AllFlightsValues (MATRIX rotations, MATRIXD rotations_date, MATRIX flights, MATRIXD flights_times,
MATRIXD aircraft_maintenance, MATRIXD alt_aircraft_dates)
{
	MATRIXI out;
	int i, j, temp_dep, temp_arr; //Iterators and temporary field
	int year_base = rotations_date[0][2], month_base = rotations_date[0][1], day_base = rotations_date[0][0]; //Date of
the earlier flight in "rotations"
	for (i = 0; i+1 < rotations.size(); i++) //For all flights in "rotations"
	{
		/* BEGIN - Look for the earlier day */
		if ( year_base > rotations_date[i][2] )
		{
			day_base = rotations_date[i][0];
			month_base = rotations_date[i][1];
			year_base = rotations_date[i][2];
		}
		if ( year_base == rotations_date[i][2] )
		{
			if ( month_base > rotations_date[i][1] )
			{
				day_base = rotations_date[i][0];
				month_base = rotations_date[i][1];
			}
			if ( month_base == rotations_date[i][1] )
			{
				if ( day_base > rotations_date[i][0] )
				{
```

```
                                        day_base = rotations_date[i][0];
                                }
                        }
                }
                /* END - Look for the earlier day */
        }
        /* BEGIN - Real Flights Data */
        for (i = 0; i+1 < rotations.size(); i++) //Calculating the departure time
        {
                LINEINT temp_line;
                temp_dep = 0; //Resets the temporary field
                /* BEGIN - Calculating the departure time */
                temp_dep += (rotations_date[i][2] - year_base) * MINBYYEAR; //Difference of years, in minutes
                temp_dep += (rotations_date[i][1] - month_base) * MINBYMONTH; //Difference of months, in minutes
                temp_dep += (rotations_date[i][0] - day_base) * MINBYDAY; //Difference of days, in minutes
                for (j = 0; j+1 < flights.size(); j++) //Looking for the flight number in "flights"
                {
                        if (rotations[i][0] == flights[j][0]) // When it finds the correct flight
                        {
                                temp_dep += ( flights_times[j][0] * MINBYHOUR + flights_times[j][1]);
//Difference of hours, in minutes, and minutes
                        }
                }
                temp_line.push_back (temp_dep);
                /* END - Calculating the departure time */
                temp_arr = 0; //Resets the temporary field
                /* BEGIN - Calculating the arrival time */
                temp_arr += (rotations_date[i][2] - year_base) * MINBYYEAR; //Difference of years, in minutes
                temp_arr += (rotations_date[i][1] - month_base) * MINBYMONTH; //Difference of months, in minutes
                temp_arr += (rotations_date[i][0] - day_base) * MINBYDAY; //Difference of days, in minutes
                for (j = 0; j+1 < flights.size(); j++) //Looking for the flight number in "flights"
                {
                        if (rotations[i][0] == flights[j][0]) // When it finds the correct flight
                        {
                                temp_arr += ( flights_times[j][2] * MINBYHOUR + flights_times[j][3]); //Difference
of hours, in minutes, and minutes
                                if (flights_times[j][4] == 1)
                                {
                                        temp_arr += MINBYDAY; //Adds one day, if the flight arrives on the next
day
                                }
                        }
                }
                /* END - Calculating the arrival time */
                temp_line.push_back (temp_arr);
                temp_line.push_back (temp_arr - temp_dep); //Duration of flight
                out.push_back (temp_line);
        }
        /* END - Real Flights Data */
        /* BEGIN - Virtual Maintenance Flights Data */
        for (i = 0; i < aircraft_maintenance.size(); i++)
        {
                if (aircraft_maintenance[i][0] != 0) //Looking for the maintenances
                {
                        LINEINT temp_line;
                        temp_dep = 0; //Resets the temporary field
                        /* BEGIN - Calculating the departure time */
                        temp_dep += (aircraft_maintenance[i][2] - year_base) * MINBYYEAR; //Difference of years,
in minutes
                        temp_dep += (aircraft_maintenance[i][1] - month_base) * MINBYMONTH; //Difference of
months, in minutes
                        temp_dep += (aircraft_maintenance[i][0] - day_base) * MINBYDAY; //Difference of days, in
minutes
                        temp_dep += aircraft_maintenance[i][3] * MINBYHOUR; //Difference of hours, in minutes
                        temp_dep += aircraft_maintenance[i][4]; //Difference minutes
                        temp_line.push_back (temp_dep);
```

```
                        /* END - Calculating the departure time */
                        temp_arr = 0; //Resets the temporary field
                        /* BEGIN - Calculating the arrival time */
                        temp_arr += (aircraft_maintenance[i][7] - year_base) * MINBYYEAR; //Difference of years, in
minutes
                        temp_arr += (aircraft_maintenance[i][6] - month_base) * MINBYMONTH; //Difference of
months, in minutes
                        temp_arr += (aircraft_maintenance[i][5] - day_base) * MINBYDAY; //Difference of days, in
minutes
                        temp_arr += aircraft_maintenance[i][8] * MINBYHOUR; //Difference of hours, in minutes
                        temp_arr += aircraft_maintenance[i][9]; //Difference minutes
                        temp_line.push_back (temp_arr);
                        /* END - Calculating the arrival time */
                        temp_line.push_back (temp_arr - temp_dep); //Duration of flight
                        out.push_back (temp_line);
                }
        }
        /* END - Virtual Maintenance Flights Data */
        /* BEGIN - Virtual Breakdown Flights Data */
        for (i = 0; i < alt_aircraft_dates.size(); i++) //Looking for the breakdowns
        {
                LINEINT temp_line;
                temp_dep = 0; //Resets the temporary field
                /* BEGIN - Calculating the departure time */
                temp_dep += (alt_aircraft_dates[i][2] - year_base) * MINBYYEAR; //Difference of years, in minutes
                temp_dep += (alt_aircraft_dates[i][1] - month_base) * MINBYMONTH; //Difference of months, in
minutes
                temp_dep += (alt_aircraft_dates[i][0] - day_base) * MINBYDAY; //Difference of days, in minutes
                temp_dep += alt_aircraft_dates[i][3] * MINBYHOUR; //Difference of hours, in minutes
                temp_dep += alt_aircraft_dates[i][4]; //Difference minutes
                temp_line.push_back (temp_dep);
                /* END - Calculating the departure time */
                temp_arr = 0; //Resets the temporary field
                /* BEGIN - Calculating the arrival time */
                temp_arr += (alt_aircraft_dates[i][7] - year_base) * MINBYYEAR; //Difference of years, in minutes
                temp_arr += (alt_aircraft_dates[i][6] - month_base) * MINBYMONTH; //Difference of months, in
minutes
                temp_arr += (alt_aircraft_dates[i][5] - day_base) * MINBYDAY; //Difference of days, in minutes
                temp_arr += alt_aircraft_dates[i][8] * MINBYHOUR; //Difference of hours, in minutes
                temp_arr += alt_aircraft_dates[i][9]; //Difference minutes
                temp_line.push_back (temp_arr);
                /* END - Calculating the arrival time */
                temp_line.push_back (temp_arr - temp_dep); //Duration of flight
                out.push_back (temp_line);
        }
        /* END - Virtual Breakdown Flights Data */
        return out;
}

void TimeLogFile (string instance, time_t start, time_t read, time_t creation, time_t model, time_t end)
{
        string line;
        LINE doc;
        ostringstream diff1, diff2, diff3, diff4, diff5;
        if ( instance == "A01")
        {
                ofstream clear("Time Log File.txt"); //Clear the file
                clear << "Instance " << instance << endl << endl; //Write the name of the Instance
                clear << "Read Time:     " << difftime(read, start) << "s" << endl; //Write the read time
                clear << "Creation Time: " << difftime(creation, read) << "s" << endl; //Write the creation time
                clear << "Model Time:    " << difftime(model, creation) << "s" << endl; //Write the model time
                clear << "Solver Time:   " << difftime(end, model) << "s" << endl; //Write the solver time
                clear << "TOTAL TIME:    " << difftime(end, start) << "s" << endl; //Write the total time
                clear <<"_____" << endl << endl;
                clear.close();
        }
```

```
	else
	{
		ifstream in("Time Log File.txt"); //Open the file
		while( getline(in,line) ) //While there's no empty line
		{
			doc.push_back (line);
		}
		in.close();
		line = "Instance " + instance + "\n"; //Write the name of the Instance
		doc.push_back (line);
		//doc.push_back ("\n");
		//doc.push_back ("\n");
		/* BEGIN - Write the read time */
		diff1 << difftime(read, start);
		line = "Read Time:     " + diff1.str() + "s";
		doc.push_back (line);
		/* END - Write the read time */
		/* BEGIN - Write the creation time */
		diff2 << difftime(creation, read);
		line = "Creation Time: " + diff2.str() + "s";
		doc.push_back (line);
		/* END - Write the creation time */
		/* BEGIN - Write the model time */
		diff3 << difftime(model, creation);
		line = "Model Time:    " + diff3.str() + "s";
		doc.push_back (line);
		/* END - Write the model time */
		/* BEGIN - Write the solver time */
		diff4 << difftime(end, model);
		line = "Solver Time:   " + diff4.str() + "s";
		doc.push_back (line);
		/* END - Write the solver time */
		/* BEGIN - Write the total time */
		diff5 << difftime(end, start);
		line = "TOTAL TIME:    " + diff5.str() + "s" + "\n_____\n";
		doc.push_back (line);
		/* END - Write the total time */
		/* BEGIN - Updating the file */
		ofstream out("Time Log File.txt"); //Creates the file
		for (int i = 0; i < doc.size(); i++)
		{
			out << doc[i] << endl;
		}
		/* END - Updating the file */
		out.close();
	}
}

//Function to get the time slot of a time, if it's not an integer, it returns de first bigger
int GetFlightTimeSlot (int dep)
{
	float temp = ((float) dep)/60;
	int i;
	for (i = 0; i < 240*60; i++)
	{
		if (i > temp)
		{
			return ((i-1) % 24);
		}
	}
}

// Function to get the flight number, as integers, in the MATRIX rotations
LINEDOUBLE RotationsFlightNum (MATRIX rotations)
{
	LINEDOUBLE out_line; //Vector of integers that will be returned
```

```
	int i, temp; //Iterator and temporary field
	/* BEGIN - conversion to integers */
	for (i = 0; i+1 < rotations.size(); i++)
	{
		temp = atof (rotations[i][0].c_str());
		out_line.push_back (temp);
	}
	/* END - conversion to integers */
	return out_line; //Return of the vector of integers
}

/* BEGINNING OF MAIN * BEGINNING OF MAIN * BEGINNING OF MAIN * BEGINNING OF MAIN *
BEGINNING OF MAIN * BEGINNING OF MAIN */

int main(int argc, char** argv)
{
	time_t start_time, read_time, creation_time, model_time, solver_time, end_time;
	//Start of the running time record
	time (&start_time);

cout << "Instance " << argv[1] << endl << endl;

	//Creation of a matrix to each given type of file
	MATRIX aircraft, airports, alt_aircraft, alt_airports, alt_flights, config, dist, flights, itineraries, position, rotations;

	//Reading of the files and storing in the respectives matrices
	aircraft = ReadIt (argv[1], "aircraft.csv");
	airports = ReadIt (argv[1], "airports.csv");
	alt_aircraft = ReadIt (argv[1], "alt_aircraft.csv");
	alt_airports = ReadIt (argv[1], "alt_airports.csv");
	alt_flights = ReadIt (argv[1], "alt_flights.csv");
	config = ReadIt (argv[1], "config.csv");
	dist = ReadIt (argv[1], "dist.csv");
	flights = ReadIt (argv[1], "flights.csv");
	itineraries = ReadIt (argv[1], "itineraries.csv");
	position = ReadIt (argv[1], "position.csv");
	rotations = ReadIt (argv[1], "rotations.csv");

	//End of the reading time record
	time (&read_time);

	//Extraction values in "config.csv"
	LINEINT config_rec_period = RecPeriod (config);
	LINEDOUBLE config_cost_delay = ConfigDelay (config);
	LINEDOUBLE config_cost_cancel_out = ConfigOutbound (config);
	LINEDOUBLE config_cost_cancel_in = ConfigInbound (config);
	LINEDOUBLE config_cost_downgrading = ConfigDowngr (config);
	LINEDOUBLE config_cost_noncompl = ConfigNonCompl (config);
	LINEDOUBLE config_weights = ConfigWeights (config);
	//Extraction values in "airports.csv"
	MATRIXD airports_values = AirportsValues (airports);
	MATRIXD airports_dep_cap_per_ts = AirportsDepCap (airports_values);
	MATRIXD airports_arr_cap_per_ts = AirportsArrCap (airports_values);
	//Extraction values in "dist.csv"
	LINEDOUBLE dist_flight_time = DistTime (dist);
	//Extraction values in "flights.csv"
	MATRIXD flights_times = FlightsTimes (flights);
	//Extraction values in "aircraft.csv"
	MATRIXD aircraft_config = AircraftConfig (aircraft);
	LINEDOUBLE aircraft_range = AircraftRange (aircraft);
	LINEDOUBLE aircraft_cost = AircraftHCost (aircraft);
	LINEDOUBLE aircraft_turnround = AircraftTurnRound (aircraft);
	LINEDOUBLE aircraft_transit = AircraftTransit (aircraft);
	MATRIX aircraft_maint = AircraftMaint (aircraft);
	MATRIXD aircraft_maintenance = AircraftMaintenance (aircraft_maint);
```

```
		//Extraction values in "rotations.csv"
		MATRIXD rotations_date = RotationsDate (rotations);
		LINEDOUBLE rotations_flight_num = RotationsFlightNum (rotations);
		//Extraction values in "itineraries.csv"
		LINEINT itineraries_id_number = ItineraryIDN (itineraries);
		LINEDOUBLE itineraries_price = ItineraryPrice (itineraries);
		LINEINT itineraries_passengers = ItineraryPassengers (itineraries);
		MATRIXD itineraries_flights_dates = ItineraryFlightsDates (itineraries);
		//Extraction values in "position.csv"
		MATRIX position2 = PositionRearrange (position);
		MATRIXD position_cabin_qty = Position2Cabin (position2);
		//Extraction values in "alt_flights.csv"
		MATRIXD alt_flights_all = AltFlightsAll (alt_flights);
		//Extraction values in "alt_aircraft.csv"
		MATRIXD alt_aircraft_dates = AltAircraftDates (alt_aircraft);
		//Extraction values in "alt_airports.csv"
		MATRIXD alt_airports_all = AltAirportsAll (alt_airports);
		//Construction of the matrix of All Flights (String)
		MATRIX all_flights_string = AllFlightsString (rotations, flights, aircraft, aircraft_maint, alt_aircraft);
		MATRIXI all_flights_values = AllFlightsValues (rotations, rotations_date, flights, flights_times,
aircraft_maintenance, alt_aircraft_dates);

		//End of creation of support variables time record
		time (&creation_time);

		//Printing one matrix of strings on the screen
		//PrintIt (airports);
/*		for (int i = all_flights_values.size()-30; i < all_flights_values.size(); i++)
		{
			cout << i << " " << all_flights_string[i][0] << " - Departure: " << all_flights_values[i][0] << endl;
			cout << i << " " << all_flights_string[i][0] << " - Arrival: " << all_flights_values[i][1] << endl;
			//cout << i << endl << rotations[i][2] << endl;
		}
*/
		//Printing one matrix of doubles on the screen
		//PrintItD (airports_values);
		//PrintItD (airports_arr_cap_per_ts);SS

		//Conversion back to a vector of strings and updating of the matrix
		////aircraft = PutInMat (aircraft, aircraft_cap, 4);


		IloEnv env;
		try
		{
			//MODEL
			IloModel model(env);

			//DECLARATIONS
			IloInt n, j, k, l, it, fl, fl2, temp, test1, test2, tempTF;
			IloInt n_flights = all_flights_string.size(); //Number of flights (real + virtual)
			IloInt n_aircrafts = aircraft.size()-1; //Number of aircrafts
		IloInt n_aircraft = aircraft.size()-1; //Number of aircrafts
		IloInt n_itineraries = itineraries.size()-1; //Number of itineraries
			IloIntArray n_time_slots (env, 24, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
22, 23); //Array of time slots, where the value is the start hour ("6" is the time slot from 06:00 to 07:00, for example)
			IloIntArray dep_times (env, n = 0); //Departure times, in minutes (Minute "0" is midnight of the day of
the first flight)
			IloIntArray arr_times (env, n = 0); //Arrival times, in minutes (Minute "0" is midnight of the day of the
first flight)
			IloIntArray d (env, n = 0); //Duration of flight 'j', in minutes
//not necess...		IloIntArray flight_dep_time_slot (env, n = 0); //Departure Time Slot Lower Bound of flight 'j'
//not necess...		IloIntArray flight_arr_time_slot (env, n = 0); //Arrival Time Slot Lower Bound of flight 'j'
			IloIntArray dep_arr_ub_array (env, n = 0); //End of the recovery period, in minutes. Array just to ajust the
upper bound of the virtul flights
```

```
		IloInt recovery_begin = GetLowerBound (rotations_date, config_rec_period); //Beginning of the recovery
period, in minutes (Minute "0" is midnight of the day of the first flight)
		IloInt dep_arr_ub = GetUpperBound (rotations_date, config_rec_period); //End of the recovery period, in
minutes (Minute "0" is midnight of the day of the first flight)
//not necess...		IloInt time_slot_ub = GetFlightTimeSlot (dep_arr_ub); //Time slot of the end of the recovery
period
		for (it = 0; it < n_flights; it++)
		{
			dep_times.add (all_flights_values[it][0]);
			arr_times.add (all_flights_values[it][1]);
			d.add (all_flights_values[it][2]);
//not necess...		flight_dep_time_slot.add (GetFlightTimeSlot (all_flights_values[it][0]));
//not necess...		flight_arr_time_slot.add (GetFlightTimeSlot (all_flights_values[it][1]));
			if (all_flights_string[it][0] == "Real Flight")
			{
				dep_arr_ub_array.add (dep_arr_ub);
cout << "(if) * " << all_flights_string[it][0] << ": " << it << " -> UB:"		<< dep_arr_ub_array[it] << endl;
			}
			else
			{
				dep_arr_ub_array.add (all_flights_values[it][1]);
cout << "(else) * " << all_flights_string[it][0] << ": " << it << " -> UB:"		<< dep_arr_ub_array[it] << endl;
			}
		}
//cout << endl << "Press ENTER to continue...";SS

		//DECISION VARIABLES

		IloIntVarArray varAF (env, n_flights, 0, n_aircraft);//Aircraft for flight 'j' | Dom(varAF) = A | '0'
represents cancellation of the flight
		IloIntVarArray varDepTimeF (env, dep_times, dep_arr_ub_array); //Departure time of flight 'j' [minutes,
starting at midnight of the day of the first flight] | Dom(varDepTimeF) = [deptime Fj0, endrecperiod]
		IloIntVarArray varArrTimeF (env, arr_times, dep_arr_ub_array); //Arrival time of flight 'j' [minutes,
starting at midnight of the day of the first flight] | Dom(varArrTimeF) = [arrtime Fj0, endrecperiod]
		IloIntVarArray varSuccAF (env, n_flights, -(n_flights), n_flights); //Successor flight after 'j' with the
same aircraft | Dom(varSuccAF = F
		IloIntVarArray varPredAF (env, n_flights, -(n_flights), n_flights); //Predecessor flight before 'j' with the
same aircraft | Dom(varSuccAF = F
		IloIntVarArray varCancelF (env, n_flights, 0, 1); //1 if flight 'j' is canceled, '0' otherwise |
Dom(varCancelF) = {0, 1}
		IloIntVarArray varCancelI (env, n_itineraries, 0, 1); //1 if itinerary 'k' is canceled, '0' otherwise |
Dom(varCancelI) = {0, 1}
		IloIntVarArray varDepTF (env, n_flights, 0, 23); //All possible values of time slot that the departure of
flight 'j' can take | Dom(varDepTF = [departure time slot, end of recovery period time slot]
		IloIntVarArray varArrTF (env, n_flights, 0, 23); //All possible values of time slot that the arrival of flight
'j' can take | Dom(varDepTF = [arrival time slot, end of recovery period time slot]
		//IloIntVarArray varValidC (env, ******** , 0, 1); //1 if the sequence 'c' is valid, '0' otherwise |
Dom(varValidC) = {0, 1}
		IloArray <IloIntVarArray> deps_per_ts (env, airports_dep_cap_per_ts.size()); //Variable to count the
number of departures per time slot
		IloArray <IloIntVarArray> arrs_per_ts  (env, airports_arr_cap_per_ts.size()); //Variable to count the
number of arrivals per time slot
		for (it = 0; it < airports_dep_cap_per_ts.size(); it++)
		{
			deps_per_ts[it] = IloIntVarArray (env, 200, 0, 1000); //Variable to count the number of
departures per time slot
			arrs_per_ts[it] = IloIntVarArray (env, 200, 0, 1000); //Variable to count the number of arrivals
per time slot
		}


		//CONSTRAINTS

		for (j = 0; j < n_flights; j++)
		{
```

```
                        if (dep_times[j] < recovery_begin || all_flights_string[j][0] == "Virtual Maintenance Flight" ||
all_flights_string[j][0] == "Virtual Breakdown Flight")
                        {
                                test1 = 0;
                                for (k = 0; k < alt_flights_all.size(); k++)
                                {
                                        if (j == GetIndexFlight (rotations, alt_flights_all[k][0]))
                                        {
cout << "HERE! j=" << j << " - k=" << k << endl;
                                                test1 = 1;
                                                test2 = k;
                                        }
                                }
                                if (test1 == 0)
                                {
cout << "fixed flights - j=" << j << endl;
                                                model.add (varDepTimeF[j] == dep_times[j]); //Constraint #17
                                                model.add (varArrTimeF[j] == arr_times[j]); //Constraint #18
                                                model.add (varAF[j] == GetIndexAircraft (aircraft,
all_flights_string[j][1])); //Constraint #12 and #13 (Fix the aircraft assigned to the flights before the recovery window)
                                }
                                else
                                {
                                        if (alt_flights_all[test2][4] == -1)
                                        {
cout << "alt flights cancel - j=" << j << endl;
                                                temp = GetIndexFlight (rotations, alt_flights_all[test2][0]);
                                                model.add (varCancelF[temp] == 1);
                                        }
                                        else
                                        {
cout << "flight:" << alt_flights_all[test2][0] << endl;
                                                temp = GetIndexFlight (rotations, alt_flights_all[test2][0]);
                                                model.add (varDepTimeF[temp] == dep_times[temp] +
alt_flights_all[test2][4]); //Constraint #15
                                                model.add (varArrTimeF[temp] == arr_times[temp] +
alt_flights_all[test2][4]); //Constraint #14
                                                model.add (varAF[temp] == GetIndexAircraft (aircraft,
all_flights_string[temp][1])); //Fix the aircraft assigned to the flights before the recovery window
cout << "alt flights - j=" << j << " - flight " <<  temp << " - delay of " << alt_flights_all[test2][4] << endl;
                                        }
                                }
                        }
                        else
                        {
                                test1 = 0;
                                for (k = 0; k < alt_flights_all.size(); k++)
                                {
                                        if (j == GetIndexFlight (rotations, alt_flights_all[k][0]))
                                        {
cout << "[N] HERE! j=" << j << " - k=" << k << endl;
                                                test1 = 1;
                                                test2 = k;
                                        }
                                }
                                if (test1 == 1)
                                {
                                        if (alt_flights_all[test2][4] == -1)
                                        {
cout << "[N] alt flights cancel - j=" << j << endl;
                                                temp = GetIndexFlight (rotations, alt_flights_all[test2][0]);
                                                model.add (varCancelF[temp] == 1);
                                        }
                                        else
                                        {
cout << "[N] flight:" << alt_flights_all[test2][0] << endl;
```

```
                                                temp = GetIndexFlight (rotations, alt_flights_all[test2][0]);
                                                model.add (varDepTimeF[temp] == dep_times[temp] +
alt_flights_all[test2][4]); //Constraint #15
                                                model.add (varArrTimeF[temp] == arr_times[temp] +
alt_flights_all[test2][4]); //Constraint #14
                                                model.add (varAF[temp] == GetIndexAircraft (aircraft,
all_flights_string[temp][1])); //Fix the aircraft assigned to the flights before the recovery window
cout << "[N] alt flights - j=" << j << " - flight " <<  temp << " - delay of " << alt_flights_all[test2][4] << endl;
                                        }
                                }
                        }
                        model.add (varDepTimeF[j] >= dep_times[j]); //Constraint #16
//                      model.add (varDepTimeF[j] >= varDepTF[j]*60); //TEST
                        model.add (varDepTF[j] != GetFlightTimeSlot(dep_times[j])); //TEST
                        model.add (varArrTF[j] != GetFlightTimeSlot(arr_times[j])); //TEST
                        model.add (varArrTimeF[j] == varDepTimeF[j] + d[j]); //Constraint #1
                        model.add (IloIfThen (env, varAF[j] == 0, varCancelF[j] == 1)); //Constraint #2
                        model.add (IloIfThen (env, varCancelF[j] == 1, varAF[j] == 0)); //Constraint #2
                        model.add (IloIfThen (env, varAF[j] > 0, varCancelF[j] == 0)); //Constraint #3
                        model.add (IloIfThen (env, varCancelF[j] == 0, varAF[j] > 0)); //Constraint #3
                        model.add (IloIfThen (env, varCancelF[j] == 1, varDepTF[j] == 0)); //Constraint #19
                        model.add (IloIfThen (env, varCancelF[j] == 1, varArrTF[j] == 0)); //Constraint #20
                }
                for (j = 0; j < n_flights; j++)
                {
                        for (k = 0; k < 200; k++)
                        {
                                model.add (IloIfThen (env, (varDepTimeF[j]/60) < k, varDepTF[j] == k-1));
                        }
SS
                        for (k = 0; k < n_flights; k++)
                        {
/*                              for (l = 0; l < n_aircraft; l++)
                                {
                                        model.add (IloIfThen (env, (varSuccAF[j] == k && varAF[j] == l &&
varCancelF[j] == 0), varDepTimeF[k] >= (varArrTimeF[j] + aircraft_turnround[l])));  //Constraint #7
                                }
*/                              model.add (IloIfThen (env, varSuccAF[j] == k, varPredAF[k] == j)); //Constraint #4
                                model.add (IloIfThen (env, varPredAF[k] == j, varSuccAF[j] == k)); //Constraint #4
                                model.add (IloIfThen (env, varSuccAF[j] == k, varAF[j] == varAF[k])); //Constraint
#8
                                model.add (IloIfThen (env, varAF[j] != varAF[k], varSuccAF[j] != k)); //Constraint
#9
                                if (all_flights_string[j][3] != all_flights_string[k][2] || arr_times[j] >= dep_times[k])
                                {
                                        model.add (varSuccAF[j] != k); //Constraint #10
                                }
                        }
                        fl = GetIndexRotFl (rotations_flight_num[j], flights);
                        if (j < rotations_flight_num.size() && flights_times[fl][6] > 0)
                        {
                                fl2 = GetIndexFlight2 (rotations, flights[fl-1][0]);
                                model.add (varSuccAF[fl2] == j); //Contraint #11
cout << "Obligatory Multileg -> j=" << j << " and fl2=" << fl2 << endl;
cout << flights[fl-1][0] << " followed by " << flights[fl][0] << endl;
                        }
                }
//              model.add (IloAllDiff (env, varSuccAF)); //Constraint #5
//              model.add (IloAllDiff (env, varPredAF)); //Constraint #6
                model.add (deps_per_ts[0][0] < 100); //TEST
                model.add (arrs_per_ts[0][0] < 100); //TEST
                model.add (IloDistribute (env, deps_per_ts[0], n_time_slots, varDepTF)); //Counting the departures per
time slot
                model.add (IloDistribute (env, arrs_per_ts[0], n_time_slots, varArrTF)); //Counting the arrivals per time
slot
//              model.add (IloMinimize(env, IloSum(varCancelF))); //"Objective Function", just to guide the search
```

```
//                      model.add (IloMaximize(env, IloSum(varSuccAF) + IloSum(varPredAF))); //"Objective Function", just
to guide the search
/*                      IloInt nbMin = 7;
                        IloInt nbMax = 30;
                        IloInt seqWidth = 5;
                        model.add(IloSequence(env, nbMin, nbMax, seqWidth, varDepTF, n_time_slots, deps_per_ts));
                        model.add(IloSequence(env, nbMin, nbMax, seqWidth, varArrTF, n_time_slots, arrs_per_ts));
*/

                        //End of model time record
cout << endl << "SOLVING..." << endl;
                        time (&model_time);

                        IloSolver solver(model);
                        if (solver.solve())
                        {
                                solver.out() << solver.getStatus() << " Solution" << endl << endl;
                                for (j = 200; j < 260; j++)
                                {
                                        if (solver.getValue(varCancelF[j]) == 0)
                                        {
                                                solver.out() << j << " * " << all_flights_string[j][0] << " - NOT
CANCELED" << endl;
                                        }
                                        else
                                        {
                                                solver.out() << j << " * " << all_flights_string[j][0] << " - CANCELED" <<
endl;
                                        }
                                        solver.out() << j << " * Aircraft: " << aircraft[solver.getValue(varAF[j])][0] << endl;
                                        solver.out() << j << " * " << "Departure: " << dep_times[j] << " -> " <<
solver.getValue(varDepTimeF[j]) << endl;
                                        solver.out() << j << " * " << "Arrival:   " << arr_times[j] << " -> " <<
solver.getValue(varArrTimeF[j]) << endl;
                                        solver.out() << j << " * " << "Departure Time Slot: " <<
GetFlightTimeSlot(dep_times[j]) << " -> " << solver.getValue(varDepTF[j]) << endl;
                                        solver.out() << j << " * " << "Arrival Time Slot:   " <<
GetFlightTimeSlot(arr_times[j]) << " -> " << solver.getValue(varArrTF[j]) << endl;
                                        solver.out() << j << " * Successor: " << solver.getValue(varSuccAF[j]) << endl;
                                        solver.out() << j << " * Predecessor: " << solver.getValue(varPredAF[j]) << endl <<
endl;
                                }
                                for (j = 0; j < 24; j++)
                                {
                                        solver.out() << "Time Slot: " << n_time_slots[j] << endl;
                                        solver.out() << "* " << solver.getValue(deps_per_ts[0][j]) << " departures" << endl;
                                        solver.out() << "* " << solver.getValue(arrs_per_ts[0][j]) << " arrivals" << endl <<
endl;
                                }
                                for (j = 250; j < 260; j++)
                                {
                                        solver.out() << "DepTime: " << solver.getValue(varDepTimeF[j]) << endl;
                                        solver.out() << "* DIV 60 = " << solver.getValue(varDepTimeF[j])/60 << endl;
                                }
                        }
                        else
                        {
                                cout << "No Solution" << endl;
                        }
                        cout << endl << endl;
                        solver.printInformation();
                }
                catch (IloException& ex)
                {
                        cout << "Error: " << ex << endl;
                }
```

```
        env.end();

        //End of the running time record
        time (&end_time);
        //Writing the recorded times in the "Time Log File"
        TimeLogFile (argv[1], start_time, read_time, creation_time, model_time, end_time);


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

        //Just to show that the program worked fine until it's end
        cout << endl << "<< by Franco Yassoyama >>" << endl;
        //Return of function main ()
        return 0;
}

/* END OF MAIN * END OF MAIN * END OF MAIN * END OF MAIN * END OF MAIN * END OF MAIN *
END OF MAIN * END OF MAIN */



/*****************************************************************************************************
*****************************************/
/*                                                DESCRIPTION OF ALL FILES
                                                */
/*****************************************************************************************************
*****************************************/


/*
Configuration File -> "config.csv"

        Recovery Period
                *** SLOT ***                    *** DATA ***
                config_rec_period[0]            Start Day
                config_rec_period[1]            Start Month
                config_rec_period[2]            Start Year
                config_rec_period[3]            Start Hour
                config_rec_period[4]            Start Minute
                config_rec_period[5]            End Day
                config_rec_period[6]            End Month
                config_rec_period[7]            End Year
                config_rec_period[8]            End Hour
                config_rec_period[9]            End Minute

        Costs associated with delay [EUR/min]
                *** SLOT ***                    *** CLASS ***           *** FLIGHT ***
                config_cost_delay[0]            First                   Domestic
                config_cost_delay[1]            First                   Continental
                config_cost_delay[2]            First                   Intercontinental
                config_cost_delay[3]            Business        Domestic
                config_cost_delay[4]            Business        Continental
                config_cost_delay[5]            Business        Intercontinental
                config_cost_delay[6]            Economy                 Domestic
                config_cost_delay[7]            Economy                 Continental
                config_cost_delay[8]            Economy                 Intercontinental

        Costs associated with cancellation on the outbound portion of a trip [EUR]
                *** SLOT ***                    *** CLASS ***           *** FLIGHT ***
                config_cost_cancel_out[0]   First                       Domestic
                config_cost_cancel_out[1]   First                       Continental
                config_cost_cancel_out[2]   First                       Intercontinental
                config_cost_cancel_out[3]   Business        Domestic
                config_cost_cancel_out[4]   Business        Continental
```

config_cost_cancel_out[5] Business Intercontinental
config_cost_cancel_out[6] Economy Domestic
config_cost_cancel_out[7] Economy Continental
config_cost_cancel_out[8] Economy Intercontinental

Costs associated with cancellation on the inbound portion of a trip or connecting passengers who have already started their trip [EUR]

*** SLOT *** *** CLASS *** *** FLIGHT ***
config_cost_cancel_in[0] First Domestic
config_cost_cancel_in[1] First Continental
config_cost_cancel_in[2] First Intercontinental
config_cost_cancel_in[3] Business Domestic
config_cost_cancel_in[4] Business Continental
config_cost_cancel_in[5] Business Intercontinental
config_cost_cancel_in[6] Economy Domestic
config_cost_cancel_in[7] Economy Continental
config_cost_cancel_in[8] Economy Intercontinental

Costs associated with downgrading [EUR]

*** SLOT *** *** FROM CLASS *** *** TO CLASS *** *** FLIGHT ***
config_cost_downgrading[0] First Business Domestic
config_cost_downgrading[1] First Business Continental
config_cost_downgrading[2] First Business Intercontinental
config_cost_downgrading[3] First Economy Domestic
config_cost_downgrading[4] First Economy Continental
config_cost_downgrading[5] First Economy Intercontinental

config_cost_downgrading[6] Business Economy Domestic
config_cost_downgrading[7] Business Economy Continental
config_cost_downgrading[8] Business Economy Intercontinental

Decreasing penalty costs associated with non-compliant location of an aircraft [EUR]

*** SLOT *** *** MISSING AIRCRAFT OF A GIVEN ***
config_cost_noncompl[0] Family
config_cost_noncompl[1] Model
config_cost_noncompl[2] Configuration

Weights alfa, beta and gama

*** SLOT *** *** WEIGHT ***
config_weights[0] Alfa
config_weights[1] Beta
config_weights[2] Gama

*/

/*
Airport Capacity File -> "airports.csv"

*** SLOT *** *** DATA ***
airports[i][0] Airport
airports[i][1] Maximum Departures/h
airports[i][2] Maximum Arrivals/h
airports[i][3] Start Time
airports[i][4] End Time
airports[i][5] Maximum Departures/h
airports[i][6] Maximum Arrivals/h
airports[i][7] Start Time
airports[i][8] End Time
... ...

*/

/*
Distance File -> "dist.csv"

*** SLOT *** *** DATA ***
dist[i][0] Origin Airport

```
                dist[i][1]              Destination Airport
                dist_flight_time[i]     Flight Time [min]
                dist[i][3]              Flight Type
*/

/*
Flight File -> "flights.csv"

                *** SLOT ***                *** DATA ***
                flights[i][0]               Number of the Flight
                flights[i][1]               Origin Airport
                flights[i][2]               Destination Airport
                flights_times[i][0]     Departure Hour
                flights_times[i][1]     Departure Minute
                flights_times[i][2]     Arrival Hour
                flights_times[i][3]     Arrival Minute
                flights_times[i][4]     '0' if Arrives on the same day; '1' is Arrives on the next day
                flights_times[i][5]     '0' if Arrives on the same day; '1' is Arrives on the next day
                flights_times[i][6]     Number of the Previous Flight; '0' if there's none
*/

/*
Aircraft File -> "aircraft.csv"

                *** SLOT ***                *** DATA ***
                aircraft[i][0]              Aircraft
                aircraft[i][1]              Model
                aircraft[i][2]              Family
                aircraft_config[i][0]       Seats Available in First Class
                aircraft_config[i][1]       Seats Available in Business Class
                aircraft_config[i][2]       Seats Available in Economy Class
                aircraft_range[i]       Range in Flight Hours [min]
                aircraft_cost[i]        Hourly Operating Cost [EUR/h]
                aircraft_turnround[i]       TurnRound Time [min]
                aircraft_transit[i]     Transit Time [min]
                aircraft[i][8]              Location at the Beginning of the Recovery Period

        Maintenance
                *** SLOT ***                *** DATA ***
                aircraft_maint[i][0]    Airport where the maintenance takes place [NULL if there's none]
                aircraft_maintenance[i][0]  Start Day
                aircraft_maintenance[i][1]  Start Month
                aircraft_maintenance[i][2]  Start Year [since 2000, i.e., 06 represents 2006]
                aircraft_maintenance[i][3]  Start Hour
                aircraft_maintenance[i][4]  Start Minute
                aircraft_maintenance[i][5]  End Day
                aircraft_maintenance[i][6]  End Month
                aircraft_maintenance[i][7]  End Year [since 2000, i.e., 06 represents 2006]
                aircraft_maintenance[i][8]  End Hour
                aircraft_maintenance[i][9]  End Minute
                aircraft_maintenance[i][10] Minutes between two consecutive Maintenance Actions
*/

/*
Rotation File -> "rotations.csv"

                *** SLOT ***                *** DATA ***
                rotations[i][0]             Flight Number
                rotations[i][2]             Aircraft
                rotations_date[i][0]    Day
                rotations_date[i][1]    Month
                rotations_date[i][2]    Year [since 2000, i.e., 06 represents 2006]
*/

/*
Itinerary File -> "itineraries.csv"
```

```
                *** SLOT ***                    *** DATA ***
                itineraries_id_number[i]        Unique Identification Number
                itineraries[i][1]               Nature of the Itinerary - Outbound (A) or Inbound (R)
                itineraries_price[i]            Average Unit Price [EUR]
                itineraries_passengers[i]       Number of Passengers on the Flight
                itineraries_flights_dates[i][0] Number of the Flight
                itineraries_flights_dates[i][1] Day of the Flight
                itineraries_flights_dates[i][2] Month of the Flight
                itineraries_flights_dates[i][3] Year of the Flight
                itineraries_flights_dates[i][4] Cabin - First (1), Business (2) or Economy (3)
                itineraries_flights_dates[i][5] Number of the Flight
                itineraries_flights_dates[i][6] Day of the Flight
                ... [until there are flights]   ... [until there are flights]
*/

/*
Aircraft Positioning Flight -> "position.csv"

                *** SLOT ***                    *** DATA ***
                position2[i][0]                 Airport
                position2[i][1]                 Aircraft
                position_cabin_qty[i][0]    Number of Seats in First Class
                position_cabin_qty[i][1]    Number of Seats in Business Class
                position_cabin_qty[i][2]    Number of Seats in Economy Class
                position_cabin_qty[i][3]    Number of Aircrafts Required
*/

/*
Flight Disruption File -> "alt_flights"

                *** SLOT ***                    *** DATA ***
                alt_flights_all[i][0]       Flight Number
                alt_flights_all[i][1]       Day of the Flight
                alt_flights_all[i][2]       Month of the Flight
                alt_flights_all[i][3]       Year of the Flight [since 2000, i.e., 06 represents 2006]
                alt_flights_all[i][4]       Delay [Minutes]; '-1' means cancellation
*/

/*
Aircraft Disruption File -> "alt_aircraft"

                *** SLOT ***                    *** DATA ***
                alt_aircraft[i][0]          Aircraft ID
                alt_aircraft_dates[i][0]    Start Day
                alt_aircraft_dates[i][1]    Start Month
                alt_aircraft_dates[i][2]    Start Year
                alt_aircraft_dates[i][3]    Start Hour
                alt_aircraft_dates[i][4]    Start Minute
                alt_aircraft_dates[i][5]    End Day
                alt_aircraft_dates[i][6]    End Month
                alt_aircraft_dates[i][7]    End Year
                alt_aircraft_dates[i][8]    End Hour
                alt_aircraft_dates[i][9]    End Minute
*/

/*
Airport Disruption File -> "alt_airports"

                *** SLOT ***                    *** DATA ***
                alt_airports[i][0]          Airport
                alt_airports_all[i][0]          Start Day
                alt_airports_all[i][1]          Start Month
                alt_airports_all[i][2]          Start Year
                alt_airports_all[i][3]          Start Hour
                alt_airports_all[i][4]          Start Minute
```

```
                alt_airports_all[i][5]          End Day
                alt_airports_all[i][6]          End Month
                alt_airports_all[i][7]          End Year
                alt_airports_all[i][8]          End Hour
                alt_airports_all[i][9]          End Minute
                alt_airports_all[i][10]         Applicable Departure Capacity, during the reduction period
                alt_airports_all[i][11]         Applicable Arrival Capacity, during the reduction period
*/

/*
Other Data

                *** SLOT ***                    *** DATA ***
                all_flights_string[i][0]    Nature of the flight (Real, Maintenance, Breakdown and, perhaps, New)
                all_flights_string[i][1]    Original Aircraft assigned for the flight
                all_flights_string[i][2]    Departure Airport of the flight ('0' for the Breakdowns)
                all_flights_string[i][3]    Arrival Airport of the flight ('0' for the Breakdowns)
                all_flights_values[i][0]    Original Departure Time of the flight [minutes]
                all_flights_values[i][1]    Original Arrival Time of the flight [minutes]
                all_flights_values[i][2]    Duration of the flight [minutes]

*/

/*****************************************************************************************************
*****************************************/
/*                                                                  END
                                                                */
/*****************************************************************************************************
*****************************************/
```