EFREI L1 Int.
Saturday, April 14, 2018

Programming Techniques and Tools
Project 2 (V_0)

Airoport Traffic Control

1 Project Presentation

The project will consist in the realization of a programme that simulate the control/management of the airport traffic. In the simplest version, the airport has a single runway and several airlines operate at the airport. The objectif is to design a program that will be able to integrate an event history module as well as a module performing the history of use of the runway in a airport.log file.

We consider that the time evolves in a discrete way : the dates are represented by an integer `t` that changes minute by minute. At each date `t`, we authorise a single event : a take off, a landing or an empty event. Each event increases the date. If there is no more waiting event, we go directly to the user choice phase. By default the date starts at 00 :00, and the process is limited to one day (23h 59mn).

1.1 Takeoffs and landings rules

In the following, the air traffic management rules :

— As much as possible, all planes fly off by the hour. Except emergency situation, a plane in the process of taking off is given priority over a plane waiting to land-wise, the second one continues to turn until the obtention of the landing authorization ;

— a plane that runs the risk of running out of fuel has priority over all other aircraft (those wishing to take off or land) ;

— a plane that belongs to a blacklisted company is no longer allowed to take off, but has priority to land on those who are not in emergency and have enough fuel.

1.2 User interaction rules

For all `t` multiples of 5, the user can perform operations among the following :

— add airplanes to take off at any time greater `t + DELAY` (the symbolic constant DELAY is 5 by default) ;

— add airplanes to the landing with any amount of fuel ;

— remove an airplane at launch ;

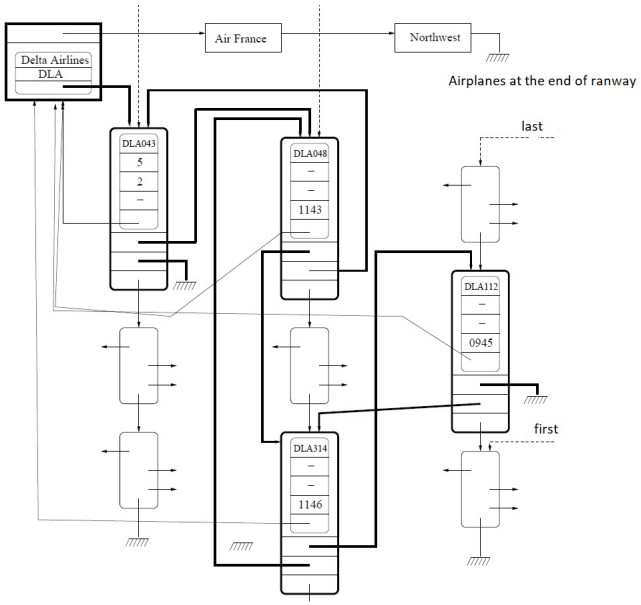
— decide that an airplane waiting for landing becomes a priority by emergency measure ; it will have to land as soon as possible, without putting the other airplanes in danger ;

— put a company on the blacklist. The programm will then have to remove all the airplanes of this company at the landing and land priority the airplanes of this company waiting for landing ;

— perform the two previous operations from a file ;

— use the random generator to perform one of the previous operations ;
— ask for the status of all the airplanes of a given company : airplanes identification of aircraft on takeoff/ landing, number of arplanes late, on time, etc . ;
— display the list of airplanes awaiting takeoff ;
— display the list of airplanes waiting to land ;
— explore the history.

2 Example of data organization



3 The used structures

In the folowing, used types are presented :
— `typedef struct avion Avion;`
— `typedef struct compagnie Compagnie;`
— `typedef struct cellule_compagnie Cellule_compagnie;`
— `typedef Cellule_compagnie *Liste_compagnie;`
— `typedef struct cellule_avion Cellule_avion;`
— `typedef Cellule_avion *Liste_avion;`

3.1 Airplanes

The planes will be represented by a structure containing these fields :
— the aiplane identifier in the form of a 6-character string (the acronym for its company on 3 characters and its flight number on 3 characters) ;
— its fuel level ;
— its fuel consumption per minute ;
— its scheduled takeoff time, in the form of a table of 4-character ;

— his company of attachment ;

```
struct avion{
char identifiant[7];
int carburant;
int consommation;
char heure_decollage[4];
Compagnie* compagnie;
};
```

3.2 The airlines

An airline is a structure containing thhese fields :

— name in full letters ;

— acronym with 3 capital letters ;

— list of its planes ;

```
struct compagnie{
char* nom;
char acronyme[3];
Liste_avions avions_compagnie;
};
```

3.3 List of planes

Each plane belongs to two lists simultaneously : one is simply chained and the other is doubly chained.

```
struct cellule_avion{
Avion avion;
struct cellule_avion* suivant_compagnie; /* pointeur sur l'avion suivant dans la liste des
avions de la compagnie */
struct cellule_avion* precedent_compagnie;/* pointeur sur l'avion pr´ec´edent dans la liste des
avions de la compagnie */
struct cellule_avion* suivant_attente; /* avion suivant dans la liste de d´ecollage ou
d'atterrissage */
}Cellule_avion;
```

When an aircraft has landed or taken off, it is removed from the lists (and the place is released).

3.4 List of the company's airplanes

The list of all the planes of a company is a doubly linked list composed by `Cellule avion` (the double chaining is insured by `suivant compagnie` and `precedent compagnie`. The field `compagnie` points to the company,

3.5 List of airplanes waiting for landing

The list of airplanes on landing is a simply linked list of cell type `Cellule avion` (the chaining is done using the `suivant attente` field). It is assumed that this list is sorted according to the number of rides used by an airplane to remain in the air (this number is calculated from its

quantity of remaining fuel and its consumption). When a new plane asks to land, it is inserted in this list in a place respecting the constraints of flight time and the schedule.

3.6 Lists of planes waiting to take off

Lists of planes waiting to Take-off management is handled by two lists. The first is a simply chained list that contains planes whose take-off time is greater than `t + DELAY`. Aircraft added to the take-off by the user or by the random generator are to be inserted in this list. The second list is of the type Tail and contains the aircrafts at the end of the runway whose time of decollage is lower than `t + DELAY` and whose order can not be modified any more. The chaining is done using the `suivant attente` fields.

```
typedef struct queue{
Liste_avions premier;/*pointe sur le premier*/
Liste_avions dernier;/*pointe sur le dernier*/
}Queue;
```

4 Coding of events

Events are encoded by a 19-character string representing the following fields separated by the '-' character :

- the airepane identifier on 6 characters ;
- a status indicator for the airplane : requests
  - A : normal landing ;
  - U : emergency landing ;
  - N : landing because the company is on the blacklist ;
  - D : takeoff ;
- the takeoff time on 4 digits or —- ;
- the quantity of fuel remaining on 2 digits or - ;
- airplane consumption per unit time on 2 digits or -.

This syntax is to be used both for entering new events and for writing the `airport.log` file and the test files. Events are indicated by a lowercase letter in `airport .log`

- a : normal landing ;
- u : emergency landing ;
- n : landing because the company is blacklisted ;
- d : take-off ;
- c : crash ;

The addition of the following 3 events :

- takeoff of Delta Airlines flight 43 ecpected for take-off at 11 :43 ;
- Approaching arrival of Northwest Airlines flight 50 with 5 fuel units and a fuel consumption of 2 per minute ;
- Alitalia 122 emergency flight arrival with 2 fuel units and consumption of 1 unit per minute ;

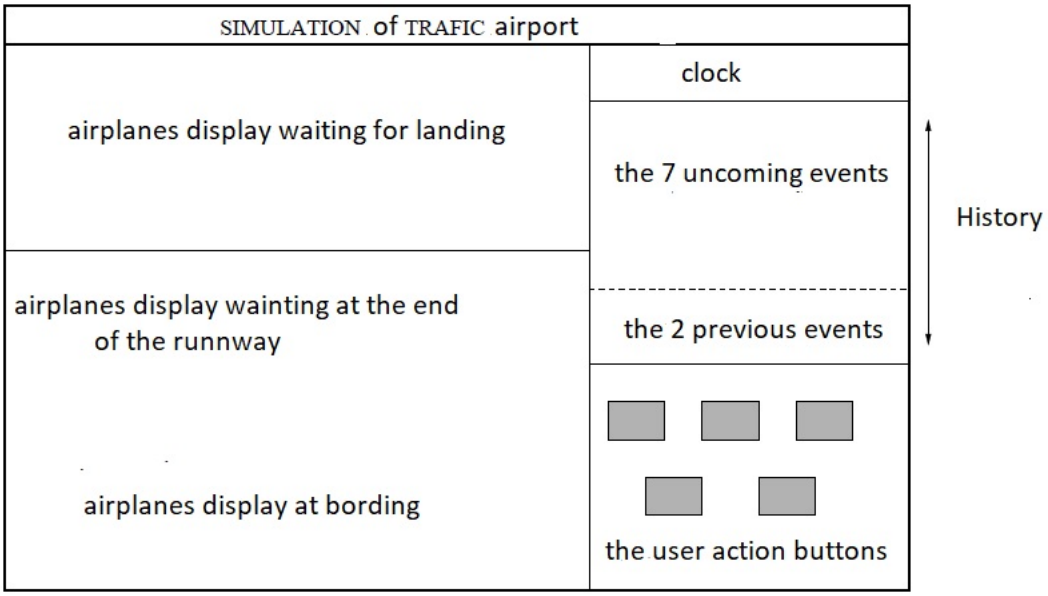
corresponds to the entry of the lines : `DLA043-D-1143 ------ NWA050 A-05-02 ------ ALI122 U-02-01 ------` An airplane exiting the waiting list due to lack of fuel will be indicated by a status indicator `c` (the crash of flight `TKC059` at `12:50` corresponds to line `TKC059-c-1250 ------` in the `airport.log` file) This same format is used in the test files to define the events to be processed. If a company's identifier is unknown, a new company is added to the list of companies. The name will be asked to the user.

5 Improvements and options

Improvements and options are not mandatory, but are a plus for the final evaluation of your work. You are invited to share your proposals with teachers to find out how relevant they really are. You will need to justify your improvements from an algorithmic point of view. They will only be taken into account if the entire project works. The following options can be considered as examples :

— include a user interface to better manage/explore the airport traffic (see the figure below)
— the assumption of a single track is very restrictive. We can consider the use of additional tracks. It is therefore important to explain your algorithmic choices. The goal is always to better manage the traffic (avoid saturations) ;
— Events could be managed globally, for example by pre-calculating future events.

s These options are given as an example. We leave you the greatest freedom in the options you want to add.



6 Schedule

The schedule of the project is as follows :

— April 14 : the project paper is submitted on `Campus Efrei`
— May 3 : Quick-Off Session. During this session, you analyze the project, divide the work to do between project members and start working on the project. If you have any question, you can ask the teachers directly. **Don't wait until this date to discover the project subject** : come with concrete questions and clear ideas about the problems you have to

face. At the end of this session, you **must** give the composition of the project teams to your teacher.

— Next, you will have three week time to work on the project at home. If you have any question, don't hesitate to send your question to your teachers.

— May 24, 4 :00 pm : Project Defense. You have 20 minutes to present your project to the teachers. Further details about the precise schedule will be given later on.

— May 24, 11 :55 pm : Project Submission. You must submit your project on Campus. You may fix bugs and/or shortcomings of your code after the defense, provided you list all the fixes in your report.

7 Deliverable

The deliverable consists of a single archive named `LASTNAME1.FirstName1.LASTNAME2.FirstName2.rar` or `.zip` to identify the project members. The archive contains :

— the report, named report.pdf

— the source files (`.c` and `.h` files only)

— a `readMe.txt` file containing a brief description of the realized part of the requirement, and the compilation/execution options.

The report is between 5 to 10 pages long. It includes an introduction, a short description of the project subject, some **algorithms** for the most important issues of the project, a discussion og the encountered problems/difficulties, and a conclusion. The report should not explain your code in details, let alone include portions of your code : good code should be self-explanatory. The code must be properly indented and commented. It must comply with the standard naming rules for variables and functions.

7.1 Project defense

Each project team has 20 minutes to defend its work, as follows :

— During the first 5 minutes, you will briefly introduce the project, then expose the results you achieved and the problems you faced. Both team members must speak in turn.

— For the next 15 minutes, you will run a demo of your program and show that you answered all (or a part of) the requirements. You have then to answer the questions of the teacher. Each team member must be able to answer any question, on any part of the project.

Please note that, depending on the quality of their answers, the project members may get different marks.

Finally, any detected plagiarism will be severely sanctioned.

Have fun