Projet Programmation 2012-2013

Cours d'informatique de Deuxième Année

—Licence L1.2—

Gestion du trafic aérien d'un aéroport

Le projet est à réaliser par binôme (2). Une soutenance aura lieu lors de la dernière séance de TP. Un rapport décrivant les fonctionnalités et justifiant le choix des méthodes devra être remis la semaine de l'avant-dernier TP. Une archive contenant l'ensemble du projet sera transmis par email à votre chargé de TP.

1 Présentation du modèle de simulation utilisé

Le projet consistera en la réalisation d'un logiciel de simulation de la gestion du trafic d'un aéroport. Dans la version la plus simple, l'aéroport dispose d'une seule piste et plusieurs compagnies aériennes exploitent l'aéroport. Votre logiciel pourra intégrer un module de génération aléatoire d'événements ainsi qu'un module effectuant l'historique d'utilisation de la piste dans un fichier `aeroport.log`.

On considère que le temps évolue de manière discrète c'est-à-dire que les dates sont représentées par un entier `t` qui évolue minute par minute. A chaque date `t`, on autorise un seul événement: un décollage, un atterrissage ou un événement vide. Chaque évènement fait augmenter la date. S'il n'y a plus d'évènement en attente, on passe directement à la phase choix de l'utilisateur. Par défaut la date commence à 0h 0mn et le déroulement est limité à une journée (23h 59mn).

1.1 Règles d'atterissage et de décollage

La gestion du trafic aérien obéit aux règles suivantes:

- autant que possible, tous les avions décollent à l'heure. Sauf situation d'urgence, un avion en instance de décollage est prioritaire sur un avion en instance d'atterrissage, le second continue à tourner jusqu'au moment où la tour de contrôle lui donne l'autorisation d'atterrissage;

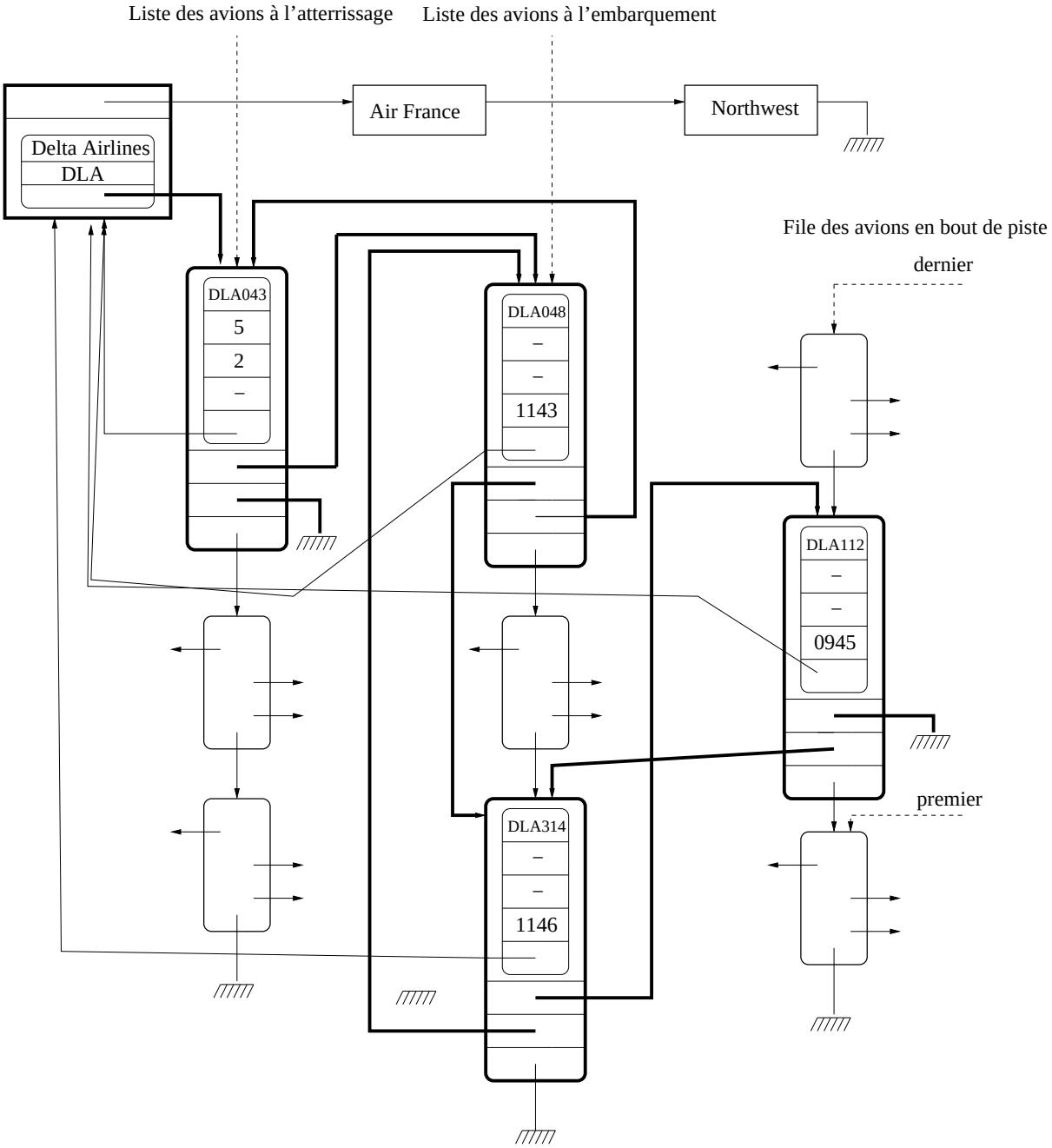
- un avion qui risque de manquer de carburant est prioritaire sur tous les autres avions (ceux qui souhaitent décoller ou atterrir);
- un avion qui appartient à une compagnie mise sur la liste noire n'a plus le droit de décoller mais est prioritaire à l'atterrisage sur ceux qui ne sont pas en situation d'urgence et qui ont suffisament de carburant.

1.2 Règles d'interaction de l'utilisateur

A toutes les dates `t` multiples de 5, l'utilisateur peut effectuer des opérations parmi les suivantes:

- ajouter des avions au décollage à des horaires quelconques supérieurs à `t+DELAI` (la constante symbolique `DELAI` vaut 5 par défaut);
- ajouter des avions à l'atterrissage avec une quantité quelconque de carburant;
- supprimer un avion au décollage;
- décider qu'un avion en attente d'atterissage devienne prioritaire par mesure d'urgence; il devra atterrir le plus tôt possible, sans mettre les autres avions en danger;
- mettre une compagnie sur la liste noire. Le logiciel devra alors retirer tous les avions au décollage de cette compagnie et faire atterrir prioritairement les avions de cette compagnie en attente d'atterissage;
- effectuer les deux opérations précédentes à partir d'un fichier;
- utiliser le générateur aléatoire pour effectuer une des opérations précédentes;
- demander le statut de tous les avions d'une compagnie donnée: identification des avions au décollage, à l'atterrissage, le nombre d'avions en retard, à l'heure, etc.;
- afficher la liste des avions en attente de décollage;
- afficher la liste des avions en attente d'atterissage;
- consultation de l'historique.

2 Exemple d'organisation des données



3 Définitions des structures utilisées

On utilise les synonymes de types suivants:

```
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 Les avions

Les avions seront représentés par une structure contenant les champs:

- l'identifiant de l'avion sous forme d'une chaîne de 6 caractères (l'acronyme de sa compagnie sur 3 caractères et son numéro de vol sur 3 chiffres);
- son niveau de carburant;
- sa consommation de carburant par minute;
- son heure de décollage prévue, sous forme d'un tableau de 4 caractères;
- sa compagnie de rattachement;

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

3.2 Les compagnies aériennes

Une compagnie aérienne est une structure contenant les champs:

- nom en toutes lettres,
- acronyme avec 3 lettres majuscules,
- liste de ses avions,

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

3.3 Liste des compagnies

Les compagnies aériennes sont stockées dans une liste simplement chaînée de cellules du type:

```
struct cellule_compagnie{
   Compagnie comp;
   struct cellule_compagnie* suivant;
};
```

3.4 Liste des avions

Chaque avion appartient simultanément à deux listes: l'une est simplement chaînée et l'autre est doublement chaînée.

```
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 précédent dans la liste des
                                              avions de la compagnie */
struct cellule_avion* suivant_attente;    /* avion suivant dans la liste de décollage ou
                                              d'atterrissage */
}Cellule_avion;
```

Lorsqu'un avion a atteri ou a décollé, il est retiré des listes (et la place est libérée).

3.4.1 Liste des avions d'une compagnie

La liste de tous les avions d'une compagnie est une liste doublement chaînée formée de cellules du type `Cellule_avion` (le double chaînage se fait à l'aide des champs `suivant_compagnie` et `precedent_compagnie`). Le champs `compagnie` pointe vers la compagnie,

3.4.2 Liste des avions en attente d'atterrissage

La liste des avions à l'atterrissage est une liste simplement chaînée de cellule du type `Cellule_avion` (le chaînage se fait à l'aide du champs `suivant_attente`). On suppose que cette liste est triée de manière croissante suivant le nombre de tours pendant lesquels un avion peut rester en l'air (ce nombre est calculé à partir de sa quantité de carburant restant et de sa consommation). Lorsqu'un nouvel avion demande à atterir, il est inseré dans cette liste à une place respectant les contraintes de temps de vol et d'horaire.

3.4.3 Listes des avions en attente de décollage

La gestion des avions au décollage se gère grâce à deux listes. La première est une liste simplement chainée qui contient les avions dont l'heure de décollage est supérieure à `t+DELAI`. Les avions ajoutés au décollage par l'utilisateur ou par le générateur aléatoire sont à insérer dans cette liste.
La deuxième liste est du type `Queue` et contient les avions en bout de piste dont l'heure de décollage est inférieure à `t+DELAI` et dont l'ordre ne peut plus être modifié. Le chaînage se fait à l'aide du champs `suivant_attente`

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

On considère qu'un avion en interdiction de vol dégage immédiatemment la piste sans avoir décollé.

4 Codage des événements

Les événements sont codés par une chaîne de 19 caractères comprenant les 5 champs suivants séparés par le caractère '`-`':

- l'identifiant de l'avion sur 6 caractères;
- un indicateur d'état pour l'avion: les demandes
  - A: atterrissage normal;
  - U: atterrissage en urgence;
  - N: atterrissage car la compagnie est sur la liste noire;
  - D: décollage;
- l'heure de décollage sur 4 chiffres ou `----`;
- la quantité de carburant restant sur 2 chiffres ou `--`;
- la consommation de l'avion par unité de temps sur 2 chiffres ou `--`.

Cette syntaxe est à utiliser aussi bien pour la saisie de nouveaux événements que pour l'écriture du fichier `aeroport.log`et des fichiers de test.
Les événements réalisés sont indiqués par une lettre minuscule dans `aeroport.log`

- a: atterrissage normal;
- u: atterrissage en urgence;

- n: atterrissage car la compagnie est sur la liste noire;
- d: décollage;
- c: crash;

L'ajout des 3 évènements suivants:

- décollage du vol Delta Airlines 43 prévu au décollage à 11h43;
- arrivée en approche du vol Northwest Airlines 50 avec 5 unité de carburant et une consommation de 2 par minute;
- arrivée en approche d'urgence du vol Alitalia 122 avec 2 unités de carburant et une consommation de 1 unité par minute;

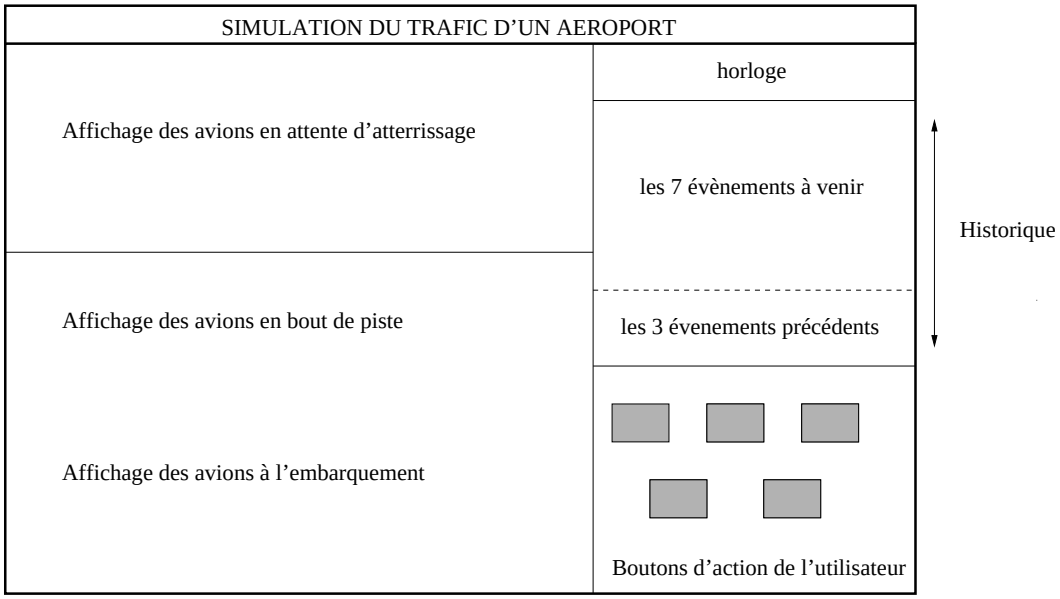
correspond à la saisie des lignes:
`DLA043-D-1143------`
`NWA050-A------05-02`
`ALI122-U------02-01`

Un avion sorti de la liste d'attente d'atterrissage pour cause de manque de carburant sera indiqué par un indicateur d'état `c` (le crash du vol TKC059 à 12h50 correspond à la ligne `TKC059-c-1250------` dans le fichier `aeroport.log`)
Ce même format est utilisé dans les fichiers tests pour définir les événements devant être traités. Si l'identifiant d'une compagnie est inconnu, une nouvelle compagnie est ajoutée à la liste des compagnies. Le nom sera demandé à l'utilisateur.

5 Interface graphique

Le logiciel sera muni d'une interface graphique programmée **exclusivement** avec la librairie graphique de l'université.

L'interface doit ressembler à:



6 Améliorations et options

Les améliorations et les options ne sont pas obligatoires, mais constituent un *plus* pour l'évaluation finale de votre travail. Vous êtes invités à faire part de vos propositions aux enseignants pour savoir dans quelle mesure elles sont réellement pertinentes. Vous devrez justifier d'un point de vue algorithmique vos améliorations. Elles ne seront prises en compte que si la totalité du projet fonctionne.

On peut envisager à titre d'exemple les options suivantes :

- l'hypothèse d'une seule piste est très restrictive. On peut envisager l'emploi de pistes supplémentaires. Il est donc important d'expliquer vos choix algorithmiques. Le but étant toujours de gérer au mieux le trafic (éviter les saturations);
- on pourrait gérer globalement les événements, par exemple en effectuant un précalcul des événements futurs.

Ces options sont données à titre d'exemple. Nous vous laissons la plus grande liberté dans les options que vous voudrez ajouter.

7 Notation

Ce projet comporte un rapport et quatre niveaux de difficulté. Il est conseillé de les réaliser dans l'ordre.

7.1 Rapport *(3pts)*

7.2 Niveau 1 *(5pts)*

- Une seule compagnie
- Gestion de listes (structure simple).
- Affichage ASCII (en mode texte) du déroulement du programme.
- Pas de collision d'horaire dans ce niveau. Tous les avions arrivent avec le carburant à la valeur maximum.
- Aucun warnings à la compilation avec les options "-Wall -ansi" de gcc.
- Respect du rendu demandé.

7.3 Niveau 2 *(4pts)*

- Exploitation des fichiers de test. Une option precisera si les évènements sont ajoutés à la liste des évènements (les évènements du fichier antérieurs à la date courante sont ignorés) ou s'ils remplace la liste des événements (toutes les listes sont vidée, la date est celle du premier ˆvènement du fichier test moins 1. Tous les évènement d'un fichier test sont ordonnés par date croissante.
- Gestion correcte de la mémoire, malloc/free.
- Gestion du fichier d'historique `aeroport.log`

7.4 Niveau 3 *(4 pts)*

- Plusieurs compagnies.
- Colision possibles des horaires et urgences de carburant possibles
- Gestion de liste avancée.
- Mise en liste noire.

7.5 Niveau 4 *(4pts)*

- Générateur aléatoire d'évenements.
- Interface graphique remplacant l'affichage ASCII.

7.6 Bonus

(Bonus quand tout le reste fonctionne)

- Interface graphique très évoluée.
- Gestion de plusieurs pistes (au moins deux). Les pistes doivent fonctionner de la même manière. Si une piste gère les urgences et l'autre les cas normaux ce niveau ne sera pas validé. Les évènement doivent être répartis sur chaque piste afin d'accellérer leur traitement.
- ...

7.7 Malus

Les malus suivants vous sont clairements indiqués et seront donc incontestables, vous êtes prévenu.

- La compilation avec `gcc -Wall -ansi` ne doit entriner ni erreurs ni `warning`.
- Non respect du mode de rendu demandé ci-dessous.
- Code mal commenté.
- Le plagiat, sous toutes ses formes, offrira un 0 au(x) groupe(s) ayant trichés.

8 Le Rendu

Ce projet est à faire en binôme. Le rendu s'effectuera par mail sous forme d'une pièce jointe au format tar.gz contenant l'ensemble des programmes et documents.
Le mail devra être envoyé à votre chargé de TP. Le fichier s'appellera *nom1_nom2.tar.gz* - nom1 et nom2 étant les noms respectifs des personnes formant le binôme, triés dans l'ordre alphabétique.

8.1 Le fichier compressé

Le *tar.gz* contiendra les répertoires et fichiers suivants:

- Un répertoire `doc` contenant votre rapport.
- Un répertoire `src` contenant le(s) fichier(s) source(s) de votre projet.
- Un fichier `README` et, pour les versions graphique, un fichier `Makefile` à la racine du répertoire.

8.2 Le README

Dans le fichier README vous expliquerez de manière claire mais sans vous attarder:

- Comment compiler et exécuter votre projet.
  Les options de compilation de gcc "-Wall -ansi" sont obligatoires, les warnings seront pénalisés.
- Les erreurs et disfonctionnements connus, tout disfonctionnement non indiqué ici sera pénalisant.
- Les améliorations possibles.

8.3 Le rapport

Votre rapport devra être au format pdf (obligatoire) et respecter les points suivants:

- Un rapport n'est pas un listing, donc n'y mettez pas votre code, nous l'avons déjà.

L'idée est de décrire le projet sous la forme Introduction, Développement, Conclusion.
L'introduction présentera le projet en lui même, sans être une recopie du sujet.
Le développement contiendra les points clefs du programme, ou problèmes rencontrés (ainsi que leur solution). Par exemple:

- Vous décrirez les principaux algorithmes utilisés.
- Une attention particulière sera portée sur vos structures de listes, expliquez-les bien.
- Vous expliquerez également comment sont lus les fichiers de test.

Enfin en conclusion, vous ferez le bilan des points forts de votre programme, et laisserez des perspectives sur les possibilités d'améliorations.