IFT1166 – TRAVAIL PRATIQUE #3 – 22 février 2004

Attachez votre ceinture! C++ est aux commandes!

Mohamed Lokbani

Équipes: Le travail est à faire en monôme ou en binôme, mais pas plus de deux. Vous ne remettez qu'un seul travail par équipe.
Remise : Deux remises à effectuer : électronique et papier le **mercredi 16 mars, 20h00 au plus tard, sans possibilité de retard**.
Conseil: N'attendez pas le dernier jour avant la remise pour commencer, vous n'aurez pas le temps.

But : Ce TP a pour but de vous faire pratiquer l'approche orientée objet. Il va vous permettre ainsi de concevoir des classes en tenant compte des différentes notions étudiées jusqu'à présent. Par ailleurs ce travail va vous permettre aussi de compléter les notions apprises dans les précédents travaux comme l'appel de fonctions externes et la compilation séparée. Il s'attardera aussi sur la façon d'utiliser les nombres aléatoires.

Énoncé : Le système automatique de contrôle de vol ("pilote automatique") est un dispositif assurant le contrôle des mouvements d'un avion dans les différentes directions. Il est activé par le pilote, le plus souvent, quand l'avion arrive à sa phase de croisière et est désactivé dans la phase de pré descente. Le pilote automatique, n'est pas une entité indépendante pouvant fonctionner toute seule, il nécessite la présence du pilote afin de modifier de temps à autre ses paramètres de navigation (altitude, cap etc.).

Durant un voyage, nous allons supposer que l'avion passe par 3 phases. Ces phases seront prises en charge entièrement par le pilote automatique et sont comme suit:

- **La phase ascendante** (Montée), l'avion prend de l'altitude jusqu'à se stabiliser à une altitude fixée.
- **La phase de croisière**, l'avion reste à un même niveau d'altitude.
- **La phase descendante** (Descente), l'avion perd de l'altitude jusqu'à se stabiliser à une altitude fixée.

Pour pouvoir réaliser ces phases, des procédures de pilotage doivent être suivies pour faire prendre (ou faire perdre) à l'avion de l'altitude. Dans notre exemple, nous avons simplifié (ou même adapté) ces procédures afin de faciliter la programmation de ce pilote automatique. Nous utiliserons la technique de paliers. Un palier est un niveau durant lequel, l'altitude et la vitesse de l'avion sont constantes.

Caractéristiques techniques : Dans ce travail pratique, vous allez concevoir les trois phases précédemment décrites et dont les caractéristiques techniques sont comme suit :

La phase de montée : L'avion monte à une altitude spécifiée préalablement. Les mouvements de montée se feront par paliers de 150 mètres chacun. Durant ces phases (en paliers), des indications sont données, sur l'altitude courante de l'avion. Le dernier palier qui sera atteint par l'avion lors de cette phase ascendante, est celui proche de l'altitude à atteindre. Par exemple, si 200 mètres est l'altitude à atteindre, alors l'avion cessera sa phase de montée à l'altitude 150 mètres (dernier palier accessible). Un avion se situant à 4000 mètres et voulant monter à 4500 mètres passera par plusieurs paliers successifs distants l'un de l'autre de 150 mètres. L'avion passera donc par les niveaux, 4150 mètres, 4300 mètres et se stabilisera à 4450 mètres le niveau le plus proche (mais inférieur) de 4500 mètres.

La phase de descente : L'avion descend à une altitude spécifie préalablement. Les mouvements de descente se feront par paliers de 150 mètres chacun, ou bien jusqu'à l'atterrissage de l'avion si l'altitude vaut 0. Durant ces phases (en paliers), des indications sont données, sur l'altitude courante de l'avion. Si à la fin de la phase de croisière l'avion se trouve à une altitude inférieure à celle demandée à la phase de descente, on forcera l'atterrissage de l'avion, car nous ne pouvons pas avoir une valeur négative du palier. Un avion qui doit passer de 4000 mètres à 3800 mètres, passera par le

niveau 3850 mètres et se stabilisera au niveau 3700 mètres le niveau le plus proche (mais inférieur) de 3800 mètres. Si le dernier niveau est de 100 mètres, et le pilote décide de positionner l'avion à 50 mètres d'altitude, on forcera l'atterrissage de l'avion, car nous ne pouvons pas avoir une valeur négative du palier (dans cet exemple, -50 mètres) .... Même s'il faut faire atterrir l'avion sur le Saint Laurent!

La phase de croisière : L'avion vole en phase de croisière pendant un temps total (T) spécifié préalablement. Il se peut que durant la phase de croisière, des perturbations viennent mettre leurs grains de sel! Ces perturbations peuvent intervenir à un intervalle de temps régulier t (où t = 0, 1, 2, ..., T ; et T la durée totale de la phase de croisière). Durant chaque instant t des choses peuvent arrivées. L'avion peut prendre de l'altitude ou en perdre, une valeur comprise entre 0 et 20 mètres. On s'assure donc que l'avion ne vole pas trop haut (ou trop bas) par rapport à l'altitude de sa phase de croisière, avec une marge d'erreur de 5 mètres. Si l'avion est trop haut on lui fait perdre un (seul) palier, s'il est trop bas, on lui fait gagner un (seul) palier. Si par exemple, l'avion se trouve à 900 mètres et nous enregistrons les perturbations suivantes aux instants t = 0 et t = 1 comme suit :

a) à t = 0, on perd 4 mètres, de ce fait l'altitude devient égale à 896, on ne fait rien ;

b) à t = 1, turbulences, montée subite de 12 mètres, altitude est dans ce cas 908 mètres, on est trop haut par rapport au seuil fixe (900 +/- 5 mètres donc pas plus de 904 mètres) => on perd de l'altitude, dans ce cas 150 mètres pour se retrouver à 758 mètres.

Simulation des perturbations : Pour simuler ces perturbations, nous utilisons des méthodes de la classe GenAleat (Elle est fournie. Voir fichier « genaleat.h »). Les méthodes de cette classe permettent de générer des nombres aléatoires et seront utiles pour faire ce qui suit:

a) on tire au hasard un nombre pour décider à un instant t quelconque ($0 \leq t < T$), s'il y a perturbation ou pas,

b) s'il y a perturbation, on tire au hasard un nombre pour décider si l'avion doit perdre ou prendre de l'altitude,

c) et finalement s'il doit perdre (ou prendre) de l'altitude, de combien de mètres? On tire au hasard cette valeur qui doit être comprise entre 0 et 20 mètres.

Travail à réaliser : Vous avez à écrire en C++ les fonctionnalités du pilote automatique. Ces fonctionnalités doivent être regroupées dans une classe appelée **Avion**, que vous devez concevoir. Par ailleurs, vous devez écrire les définitions de toutes les méthodes membres de la classe **Avion**. Les fonctionnalités doivent être représentées par les méthodes suivantes :

Phase	Méthode	Description
Montée	void Montee(int);	L'avion monte tel que l'altitude >= à la valeur de l'argument
Descente	void Descente(int);	L'avion descend tel que l'altitude <= à la valeur de l'argument
Croisière	void Croisiere(int);	Argument = T: temps total de la phase de croisière

La classe Avion et ses méthodes doivent être définies dans les fichiers « avion.h » et « avion.cpp ».

Fichiers fournis :

tp3H05.cpp : Dans la configuration proposée, le pilote sera représenté par un programme de simulation, qui est fourni. Les commandes transmises au pilote automatique seront les instructions écrites dans la fonction main de ce programme. Le fichier « tp3H05.cpp » représente ce programme.

genaleat.h : Ce fichier contient la description de la classe `GenAleat`. Cette classe permet la génération des nombres aléatoires uniformément repartis sur un intervalle donné. Les méthodes définies dans cette classe, permettent de générer un nombre aléatoire de type int ou double.

genaleat.o : Ce fichier est le résultat de la compilation de **genaleat.cpp**. Vous avez deux versions de ce fichier. Une a été compilée sur un des postes Windows (XP) de la DESI et une autre a été compilée sur un des postes Linux du département ayant comme noyau (2.6.10-1.14_FC2 i386).

resultat.txt : Ce fichier contient les résultats obtenus après exécution du programme de simulation dont le code source est **tp3H05.cpp**.

Contraintes algorithmiques : Nous n'imposons aucune approche pour résoudre ce problème.

Contraintes Techniques :
- Assurer vous de respecter les noms des fichiers demandés, les noms des méthodes ainsi que le format de l'affichage en sortie.

- Des fichiers d'entrée/sortie sont fournis. On ne vous demande pas d'écrire du code pour lire/écrire à partir de fichiers, vous pouvez tester vos programmes à la main (en insérant manuellement les informations) comme vous pouvez utiliser les redirections (plus facile à manier et évite les erreurs de frappe). Les redirections sont au nombre de deux : < pour l'entrée et > pour la sortie. Exemple : [**tp3.exe > masortie.txt**]. Dans cet exemple l'écriture se fera dans le fichier masortie.txt. Il vous est possible aussi d'utiliser des options prévues dans le fichier tp3H05.cpp. Lire le contenu de ce fichier pour avoir plus de détails.

- Les fichiers entrée/sortie ont été générés sous **MinGW/Msys,** ils sont donc au format DOS étendu. Si vous travaillez sur LINUX assurez-vous de les convertir au format LINUX en utilisant pour cela la commande **dos2unix**. Pour convertir de LINUX vers le format dos utiliser plutôt la commande **unix2dos**. Ces commandes sont disponibles dans un xterm d'une plateforme LINUX.

- À signaler que nous utiliserons d'autres fichiers pour la correction. Si vous avez respecté les contraintes imposées, peu importe le jeu de test utilisé, il devra fournir normalement un résultat correct.

Automatisation de la correction : C'est « l'ordinateur » qui va corriger en quelque sorte vos travaux. Ce dernier va utiliser des scripts qui vont tenir compte au détail près des contraintes imposées.
Rediriger la sortie dans un fichier avec le symbole > et la comparer au caractère près avec les sorties fournies en exemple, aider vous dans cette tâche de la commande diff (sous MinGW/MSys sinon la commande fc sous dos, ajuster juste les options) comme suit :

```
diff –b –w –i –B out1.txt votresortie.txt
```

Par ailleurs, nous utiliserons d'autres jeux de test pour la correction.

Si les fichiers sont identiques, vous n'obtenez rien en sortie. Sinon, vous obtiendrez les différences entre les deux fichiers. Pour plus de détails sur cette commande unix, sur votre console xterm, exécutez la commande suivante: **man diff**.

Compilation : Pour générer l'exécutable, tp3.exe, vous pouvez utiliser par exemple une des solutions suivantes :

-1- Utiliser le fichier Makefile fourni avec ce travail. Lire le contenu du fichier pour comprendre comment l'utiliser.

-2- Faire une compilation séparée à la main :
- a) compiler le fichier avion.cpp à part (ou CTRL-F7 sous scite) :
  g++ -c avion.cpp -Wall -pedantic -Os -I/. -L./
- b) compiler le fichier tp3H05.cpp à part (ou CTRL-F7 sous scite) :
  g++ -c tp3H05 -Wall -pedantic -Os -I/. -L./
- c) générer l'exécutable (ou F7 sous scite):
  g++ -o tp3.exe tp3H05.o genaleat.o avion.o -Wall -pedantic -Os -I/. -L./

tp3.exe sera le programme exécutable. Ne pas oublier de mettre dans le même répertoire les fichiers *.h et *.cpp pour que la compilation puisse avoir lieu correctement.
En utilisant scite, la touche F7 permet de construire le projet. Elle utilise pour cela le fichier Makefile s'il est présent dans le répertoire. Donc faire attention à ce détail ! Placer le fichier Makefile dans un autre répertoire si vous ne voulez pas l'utiliser.

Remise : Il est important de noter que votre TP sera compilé avec gcc3.2. Si par choix vous décidez d'utiliser un autre compilateur, vérifiez que le code que vous avez produit (qui normalement fonctionne correctement chez vous)

fonctionne bien sur les ordinateurs du DESI. Avant de faire cela, assurez-vous d'abord que vous avez la bonne version de gcc, en utilisant pour cela la commande : "**gcc –v**" devra donnée le numéro de version "**3.2**".

La remise comprend **3** (**TROIS**) choses (avant le mercredi 16 mars à 20h00) :

1. Envoyez vos fichiers « avion.cpp », « avion.h » et le rapport par la procédure de remise électronique habituelle (Pour obtenir de l'aide sur cette commande, tapez dans un xterm : man remise). Respecter les noms des fichiers.

   remise pift1166 travail03 avion.h avion.cpp rapport

2. Remettez une **copie papier** d'un rapport qui devrait décrire votre programme (pour le contenu d'un rapport voir la FAQ sur la page web du cours).

3. N'oubliez pas de remettre avec votre rapport une **copie papier** de votre programme C++.

Barème : Ce TP1 est noté sur 12 points.

Compilation et respect des spécifications	**1**
Algorithmique, codage, commentaires etc.	**4**
Rapport	**3**
Tests (fournis et non fournis)	**4**

En plus du précédent barème, vous risquez de perdre des points dans les cas suivants ….

- La non remise électronique (volontaire ou par erreur) est sanctionnée par la note 0.
- La non remise papier vous pénalise de 1 point.
- Les programmes ne contenant pas d'en-tête, -1 point.
- Un programme qui ne compile pas : 0/12.
- Un programme qui compile mais ne fait les choses prévues dans la spécification : 0/12.
- Les avertissements (warnings) non corrigés : cela dépend de la quantité! À partir de -0.5 et plus.
- Le nom respect du nom de fichier, donc forcement il ne va pas compiler et un des points de la spécification n'a pas été respecté : 0/12.
- Aberration dans le codage : Même si tous les chemins mènent à Rome, faites l'effort nécessaire pour éviter de prendre le plus long! -0.5 et plus, en fonction des dégâts constatés !

Des questions à propos de ce TP?

Une seule adresse : pift1166@iro.umontreal.ca

Pour faciliter le traitement de votre requête, inclure dans le sujet de votre email, au moins la chaîne: [IFT1166] et une référence au **travail03**.

Mise à jour

22-02-2005 diffusion

Questions et Réponses

Questions Pratiques

-Q : En quoi consiste notre travail au juste?
-R : Écrire la classe Avion et ses méthodes Montée, Descente et Croisière.

-Q : Où doit-on écrire nos programmes?
-R : Les fichiers avion.h et avion.cpp

-Q : Est-ce que je dois écrire une fonction main ?
-R : Non, puisqu'on fournit une. Voir le fichier tp3H05.cpp.

-Q : Est-ce que je peux modifier alors le contenu du fichier tp3H05.cpp ?
-R : Non vous ne pouvez pas modifier le fichier tp3H05.cpp. Ceci écrit, vous pouvez quand même écrire un pseudo fichier montp3.cpp qui va contenir votre fonction main test et ceci dans le processus de développement uniquement. Avant de remettre votre travail assurez-vous de compiler avec la version originale fournie, i.e. tp3H05.cpp. C'est cette version et rien d'autre que nous allons utiliser lors de la correction.

-Q : Comment utiliser les options prévues sur la ligne de commandes ?
-R : Ces options sont décrites dans le fichier tp3H05.cpp. Je vous suggère de vous attarder dessus pour apprendre la technique qui permet d'inclure des arguments sur la ligne de commandes.

Q : Qu'est-ce qu'il en est pour les fichiers genaleat.h et genaleat.o. Est-ce que je peux les modifier ?
R : Non, il ne vous est permis de les modifier.

-Q : Pourquoi avoir fourni le binaire de genaleat.cpp et non pas le code source ?
-R : Dans le travail 2, nous vous avons permis de regarder comment étaient codées les différentes fonctionnalités fournies même si le but était uniquement de comprendre comment les utiliser. Dans ce travail, nous retirons les informations « inutiles » et nous vous plaçons dans un cas réel.

-Q : L'énoncé n'est pas clair et j'ai besoin d'aide ! Comment procéder ?
-R : Nous sommes disponibles suivant l'horaire affiché sur la page web du cours, rubrique « Professeurs et démonstrateurs ». Vous pouvez aussi nous envoyer vos questions par courriel.

-Q : Mais je ne pourrai pas être présent aux horaires indiqués ?
-R : Envoyer nous un courriel pour voir s'il nous est possible de vous rencontrer à un autre moment.

Q : Pouvez-vous m'expliquer ce qui ne va pas? Je vous envoie pour cela mes fichiers avion.cpp et avion.h !
R : N'envoyer rien par courriel. Privilégier plutôt les séances de disponibilité offertes à vous.

-Q : Où se trouvent les fichiers « tp3H05.cpp », « genaleat.h », « genaleat.o » et « resultat.txt » ?
-R : Sur la page web du cours, rubrique « Démonstrations et devoirs », puis « tp3 ».

Questions Techniques générales

-Q : Est-ce que le pilote automatique s'occupe de toutes les phases ou uniquement la phase de croisière ?
-R : Le pilote automatique s'occupe des trois phases, montée, croisière et descente. En plus de cela il se charge de régulariser l'altitude en cas de perturbations externes.

-Q : Pourquoi vous nous parler de paliers?

-R : Vous n'êtes pas dans un avion de chasse où vous pouvez vous permettre de monter ou descendre en « flèche »! C'est après tout un avion de ligne! Attention à ne pas trop secouer sa carlingue et sa « cargaison »!

-Q : Je n'ai pas compris ce que vous voulez dire par « il vous est demandé d'écrire la définition de toutes les méthodes de la classe Avion » ?
-R : Voir les démos ou le cours pour avoir une idée que doit contenir un *.h et *.cpp

-Q : Je viens de compiler l'ensemble des fichiers avec g++. J'obtiens cette erreur que je n'arrive pas à décoder ?
g++ -Wall -o tp3.exe avion.cpp genaleat.o tp3H05.cpp

ld: fatal: file genaleat.o: wrong machine type
ld: fatal: File processing errors. No output written to tp3.exe
collect2: ld returned 1 exit status
-R : Vous utilisez le mauvais binaire. Il faudra faire attention de prendre le binaire de genaleat.o compilé sous Windows si vous êtes sur cette plate forme sinon celui de Linux si vous êtes sur cette dernière.

-Q : C'est quoi « 2.6.10-1.14_FC2 i386 »?
-R : Le noyau de Linux utilisé pour la compilation porte la version 1.6.10-1.14. Il a été installé dans un environnement Fedora Core 2, d'une machine i386. Juste pour dire que ce n'est pas parce que c'est un binaire Linux que ce binaire est accessible sur toutes les plates formes Linux. Faire attention de vous assurer que vous utilisez une version de Linux compatible. Par ailleurs, ce noyau est installé sur certaines machines Linux du département comme Phobos, Deimos etc.

-Q : Une perturbation est générée par la fonction IntAleat(). À chaque instant, la valeur est générée aléatoirement pour la période prédéfinie (ex IntAleat(2,6)). Mais, dans votre résultat, la perturbation (t = 0) se fixe à la chute (ou montée) de X mètres. Est-ce que je suis obligé de générer cette valeur pour l'instante t =0 par appeler la fonction IntAleat(X,X) ?
-R : Ce qui est aléatoire c'est ce que la fonction va choisir comme donnée à retourner suite au passage d'un argument (une valeur) qui elle n'est pas aléatoire

-Q : Le résultat de sortie après l'exécution du programme « tp3H05.cpp » devrait être semblable au résultat qui se trouve dans votre fichier « resultat.txt »? Alors comment on peut dire que la classe génère des nombres aléatoires, a moins que vous aviez défini (figé) la méthode de telle sorte que l'appel avec le paramètre (durée) donne toujours les mêmes nombres aléatoires , ceci contredit la notion de nombres aléatoires ?
-R : Si le départ est fixé peu importe le nombre de fois où il y a eu un appel à la méthode « random », on dit que cette approche est « pseudo aléatoire ». Si par contre ce point de départ, est lié à l'horloge interne du système, qui elle, n'est pas fixe, on dit que cette approche est "presque" aléatoire. L'utilisation de « presque » est du au fait à la cuisine mathématique qu'il y a derrière. Pour plus de détails sur os/Linux: « man random ». Sinon un bon livre de cryptographie. Pour ce travail, nous utilisons une approche pseudo aléatoire. Ce choix a été codé dans le fichier « genaleat.o ». Ainsi, si vous exécutez « 50 » fois le même programme, vous allez générer exactement les mêmes nombres !

-Q : Comment est-ce possible d'obtenir exactement la même sortie que celle du fichier resultat.txt, si les turbulences sont déterminés par des nombres aléatoires (plutôt pseudo aléatoires)?
-R : Il n'y a pas de magie ! Le codage de la méthode random utilisée dit que si on part d'un même point et si ce point est fixé, on doit arriver au même endroit. Comme nous avons fixé le point de départ dans le genaleat.o, forcément les sorties doivent être les mêmes.

-Q : Pourquoi y a-t-il une différence entre les résultats obtenus sous Windows et Linux ?
-R : Des systèmes différents, ayant des horloges internes différentes donc des points de départ différents. Le résultat sera forcément différent.