INF1010
Programmation Orientée-Objet

Travail pratique #5
Fonction, classe générique et Bibliothèque STL

Objectifs :	Permettre à l'étudiant de se familiariser avec les listes liées, les fonctions et classes génériques ainsi qu'avec les conteneurs et algorithmes de la bibliothèque STL.
Remise du travail :	Lundi 01 avril 2013, 10h
Références :	Notes de cours sur Moodle & Chapitres 5.9, 12, 13, 16 et 20 du livre Big C++ 2e édition
Documents à remettre :	Les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip
Directives :	Directives de remise des Travaux pratiques sur Moodle Les en-têtes et les commentaires sont obligatoires. Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. Veuillez suivre le guide de codage

---

Un réseau informatique est constitué d'un ensemble d'ordinateurs reliés et communiquant entre eux à l'aide de messages structurés. Vous aurez lors de ce TP à simuler le comportement d'un réseau simpliste basé sur l'échange de matrices de données. La structure de ce réseau, présentée à la figure 1, est quelque peu particulière. En effet, les ordinateurs ne sont pas connectés entre eux mais le sont par l'intermédiaire d'un objet *Réseau* qui assure l'envoi des messages entre les ordinateurs. Un ordinateur *A* désirant envoyer un message à l'ordinateur *B* doit demander au *Réseau* d'envoyer le message à sa

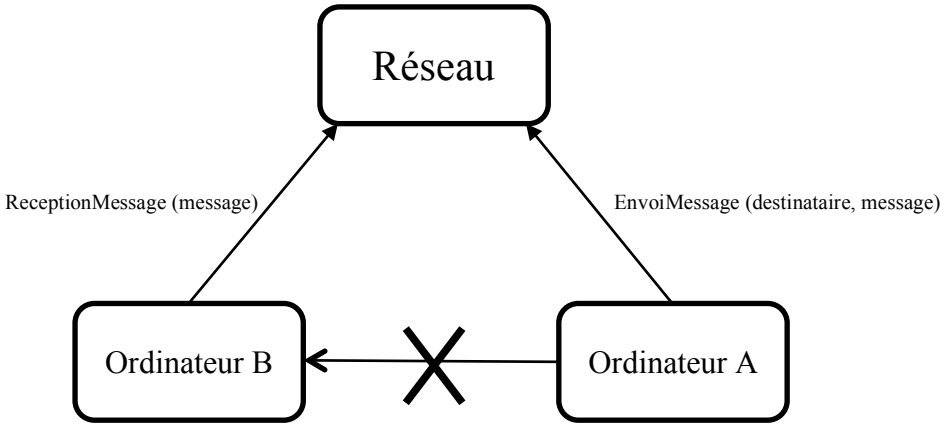

Figure 1 : Représentation simpliste du réseau

place.

Le fonctionnement du programme principal est découpé en trois phases :

1. **Construction du réseau** : construction de la liste des ordinateurs et remplissage des messages à envoyer.
2. **Fonctionnement du réseau** : envoi des messages entre les ordinateurs par l'intermédiaire du réseau.
3. **Résultats** : Affichage des messages reçus pour chaque ordinateur du réseau.

De plus, les messages qui seront transmis entre ordinateurs contiendront votre propre structure de données, implémentée au préalable.

Travail à réaliser :

(Lisez l'énoncé entièrement avant de commencer à programmer…)

Classe Matrice

La première partie du TP consiste en l'implémentation d'une classe générique représentant la structure de données qui sera échangée sur le réseau : une matrice rectangulaire de taille quelconque contenant n'importe quel type de données.

En plus d'un constructeur par défaut qui construit une matrice 3x3 vide et d'un destructeur, implémentez :

- Un constructeur par paramètres en indiquant le nombre de lignes et de colonnes ;
- Un constructeur par copie
- La surcharge de l'opérateur () permettant d'accéder et de modifier un élément en lui fournissant la ligne et la colonne de celui-ci. Vous devez vérifier la validité des paramètres en entrée;
- La surcharge de l'opérateur << pour afficher les éléments de la matrice
- La surcharge de l'opérateur >> pour remplir les éléments de la matrice.

Cette partie ne devrait pas vous prendre plus d'une séance de TP.

La deuxième partie consiste en l'implémentation du réseau. Les messages contiennent les informations brutes, c'est-à dire les matrices de données, et sont contenus au sein des ordinateurs. Ceux-ci communiquent entre eux uniquement par l'intermédiaire de l'objet *Reseau*. Il y a plusieurs possibilités

pour contenir les ordinateurs dans le réseau. Nous vous demandons d'en implémenter deux : les *list* et les *map*.

Classe *Message*

Cette classe contient les données que vous allez échanger. Nous utiliserons la classe générique ***Matrice*** que vous avez implémenté dans la première partie du TP comme contenu du message. Une matrice de taille 3x3 contiendra neuf valeurs numériques de type *float*. La classe contiendra :

- les noms de l'expéditeur et du destinataire du message ;
- le contenu du message sous forme de matrice 3x3 contenant des *float* ;
- une variable représentant la priorité du message, de type entier pouvant aller de 1 à 3.

Elle devra implémenter :

- un constructeur par défaut, un constructeur par paramètre ;
- des méthodes pour accéder aux variables « expediteur », « destinataire » et « priorite » ainsi qu'une méthode pour modifier l'expéditeur ;
- la surcharge de l'opérateur de flux << qui devra afficher le message en respectant l'affichage suivant :

```
Message de priorité [priorité] recu de [nomExpediteur] :
[Contenu]
```

Classe Ordinateur

Cette classe représente un ordinateur sur le réseau et contient cinq variables privées :

- son adresse de type *unsigned int* ;
- son nom de type *string* ;
- l'adresse (de type *Reseau**) vers l'objet *Reseau* qui le contient, afin de pouvoir lui envoyer les messages ;
- une liste de messages créés dynamiquement à envoyer contenus dans une *priority_queue*. Dans ce type de structure, les messages sont classés selon un ordre de priorité. Vous devez donc créer un foncteur permettant de comparer deux messages selon leur priorité. La syntaxe pour la création de cette variable est :

```
priority_queue<Message*, vector<Message*>, PrioriteMessages> listMessagesAEnvoyer_;
```

où *PrioriteMessages* est un foncteur.

- un vecteur contenant les messages reçus par l'ordinateur sous forme dynamique.

Vous devez aussi implémenter dans cette classe :

- un constructeur par défaut et un destructeur ;
- un constructeur par paramètre recevant le nom et l'adresse de l'ordinateur ainsi que l'adresse du réseau le contenant ;
- des méthodes permettant d'accéder aux variables de nom, d'adresse et de réseau ;
- une méthode permettant d'ajouter un message dans la liste des messages à envoyer. Cette méthode sera appelée par le réseau ;
- une méthode *envoiMessages()* permettant d'envoyer tous les messages contenus dans la liste des messages à envoyer. Cette méthode sera appelée par le réseau dans la méthode *run()* ;
- une méthode permettant de recevoir un message et l'insérant dans la liste des messages reçus. Cette méthode sera appelée par le réseau ;
- une méthode permettant d'afficher l'ensemble des messages reçus en respectant l'affichage suivant :

```
[AfficherOrdinateur] Affichage des messages reçus ([nombre de messages reçus]) :
…
…
…
```

- la surcharge de l'opérateur de flux << permettant d'afficher l'ordinateur en respectant l'affichage suivant :

```
[Nom]   #[Adresse]
```

Classe Reseau

La classe **Reseau** est une classe abstraite sans attributs reprenant l'ensemble des méthodes propres à n'importe quel type de réseau :

- Constructeur par défaut et destructeur ;
- Une méthode permettant d'ajouter un ordinateur au réseau en recevant son nom et son adresse en paramètre (voir les types dans la classe **Ordinateur**) ;
- Une méthode permettant d'ajouter un message à un ordinateur lors de la phase de construction du réseau en recevant son adresse et le message en paramètre ;
- Une méthode permettant de supprimer un ordinateur du réseau en indiquant son nom et son adresse ;
- Trois méthodes *envoiMessage()* recevant un message en paramètre et permettant 1) d'envoyer un message à un ordinateur en recevant son adresse ou 2) en recevant son nom, et 3) d'envoyer un message à l'ensemble des ordinateurs présents sur le réseau ;
- Une méthode permettant d'afficher les ordinateurs du réseau triés par nom ;

- Une méthode permettant d'afficher les ordinateurs du réseau triés par adresse ;
- Une méthode *run()* permettant de lancer le fonctionnement du réseau. Elle doit demander à tous les ordinateurs d'envoyer leurs messages préalablement enregistrés dans la phase de construction du réseau. Les messages sont contenus dans la classe **Ordinateur** ;
- Une méthode permettant d'afficher l'ensemble des messages reçus par les ordinateurs. Cette fonction est appelée lors de la phase de résultats ;
- Une méthode permettant de vider le réseau : supprimer l'ensemble des ordinateurs du réseau.

La totalité de ces méthodes sont virtuelles pures.

Classe ReseauMap

Cette classe hérite de la classe Reseau et doit implémenter l'ensemble de ses fonctions. Elle contient les ordinateurs dans deux map :

- map<nom, Ordinateur*>
- map<adresse, Ordinateur*>

L'ajout d'un ordinateur dans ce type de réseau créée dynamiquement l'ordinateur et insère son adresse dans les deux maps. Lors de l'ajout d'un message à un ordinateur, vous ne devez donc pas ajouter le message dans les deux maps puisque celles-ci contiennent les mêmes ordinateurs.

Dans cette classe, vous devez utiliser de façon appropriée l'opérateur [] et les itérateurs.

Classe ReseauListe

Cette classe hérite de la classe Reseau et doit implémenter l'ensemble de ses fonctions. Elle contient les ordinateurs dans une liste : list<Ordinateur*>.

L'ajout d'un ordinateur dans ce type de réseau créée dynamiquement l'ordinateur et insère son adresse dans la liste.

Dans cette classe, vous devez utiliser les algorithmes de la bibliothèque STL (et donc des foncteurs). Il ne doit pas y avoir de boucle dans cette classe excepté pour la méthode envoyant un message à tous les ordinateurs présents sur le réseau. Vous pouvez regrouper les foncteurs dans cette classe dans un fichier *FoncteursOrdinateur*.

Main.cpp

Le fichier main.cpp contiendra les instructions permettant de construire le réseau, de le lancer et d'en afficher les résultats. Pour construire le réseau, vous devez tout d'abord lire le fichier *Ordinateur.txt* contenant la liste des ordinateurs à insérer dans le réseau et ensuite lire le fichier *Messages.txt* contenant les messages à envoyer par chacun des ordinateurs. La syntaxe des fichiers de données est expliquée dans le fichier *main.cpp*.

Veuillez suivre les instructions fournies.

Pendant le lancement du réseau et donc l'envoi de messages, vous devez afficher l'envoi et la réception des messages par les ordinateurs. L'affichage doit être réalisé comme sur la figure 2.

```
**********************************************************************
Demarrage du reseau : envoi des messages
**********************************************************************

Envoi d'un message a l'ordinateur Skynet #3
Skynet #3 Reception d'un message...

Envoi d'un message a l'ordinateur Marvin #2
Marvin #2 Reception d'un message...

Envoi d'un message a l'ordinateur Skynet #3
Skynet #3 Reception d'un message...

Envoi d'un message a l'ordinateur K.I.T.T. #5
K.I.T.T. #5 Reception d'un message...

Envoi d'un message a l'ordinateur Marvin #2
Marvin #2 Reception d'un message...

Envoi d'un message a l'ordinateur Hal9000 #1
Hal9000 #1 Reception d'un message...

Envoi d'un message a l'ordinateur Marvin #2
Marvin #2 Reception d'un message...

Envoi d'un message a l'ordinateur Hal9000 #1
Hal9000 #1 Reception d'un message...

Envoi d'un message a tous les ordinateurs...
DeepBlue #4 Reception d'un message...
Hal9000 #1 Reception d'un message...
K.I.T.T. #5 Reception d'un message...
Marvin #2 Reception d'un message...
Skynet #3 Reception d'un message...

Envoi d'un message a l'ordinateur Marvin #2
Marvin #2 Reception d'un message...

Envoi d'un message a l'ordinateur Skynet #3
Skynet #3 Reception d'un message...

Envoi d'un message a l'ordinateur DeepBlue #4
DeepBlue #4 Reception d'un message...

Envoi d'un message a l'ordinateur Skynet #3
Skynet #3 Reception d'un message...

Envoi d'un message a l'ordinateur Hal9000 #1
Hal9000 #1 Reception d'un message...

Envoi d'un message a l'ordinateur DeepBlue #4
DeepBlue #4 Reception d'un message...

Envoi d'un message a l'ordinateur DeepBlue #4
DeepBlue #4 Reception d'un message...
```

Figure 2 : Affichage du fonctionnement du réseau

Questions

Joindre à votre archive un fichier Question.txt ou Question.pdf contenant les réponses.

1. À quoi servent les objets fonctions (foncteurs) ?
2. Si vous aviez la possibilité de choisir entre les deux méthodes (map ou list), laquelle auriez-vous choisie ? Justifiez votre réponse en donnant les avantages et inconvénients de chaque méthode.

Correction :

La correction du TP5 se fera sur 20 points. Voici les détails de la correction :

- (05 points) Compilation et exécution exactes des différentes méthodes ;
- (02 points) Implémentation de la classe générique ;
- (04 points) Implémentation de la partie « map »;
- (04 points) Implémentation de la partie « list »;
- (02 points) Respect des consignes de l'énoncé ;
- (01 points) Documentation du code ;
- (02 points) Réponses aux questions.