INF1010

Programmation Orientée-Objet

Travail pratique #5
Fonctions et classes génériques, librairies STL

Objectifs :	Permettre à l'étudiant de se familiariser avec le concept de fonctions et des classes générique ainsi qu'introduction à la librairie STL.
Remise du travail :	Mardi 22 Novembre 2016, 8h
Références :	Notes de cours sur Moodle & Chapitres 11 à 14, 16 et 20 du livre Big C++ 2e éd.
Documents à remettre :	La solution ainsi que 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 (fichiers, fonctions) 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

Informations préalables

Compilation : le cas des inclusions (ou dépendances) circulaires. On appelle inclusion circulaire le fait que deux « points .h » s'incluent mutuellement. L'usage simple des gardes de compilation ne suffit plus à protéger le programmeur d'une erreur de compilation. En effet, les gardes de compilation (***#ifndef***, ***#define*** et ***#endif***) empêchent qu'un même fichier soit inclus plusieurs fois. Mais le problème des dépendances circulaires réside dans le fait que chacun des deux fichiers a besoin des déclarations présentes dans l'autre. En particulier, si on imagine deux classes ayant chacune une méthode prenant l'autre type de classe en argument, on a alors un cas de dépendances circulaires. Pour résoudre ce problème, on utilise des déclarations anticipées. C'est-à-dire que l'on prévient le compilateur de l'existence d'une classe via la déclaration *class maClasse;* on définira la seconde classe après ces directives (dans le même fichier). Et on fera l'opération inverse dans le fichier contenant la déclaration de *maClasse*.

Notez, toutefois, que les inclusions circulaires peuvent ralentir la compilation et doivent être utilisées avec parcimonie.

Enfin, certaines fonctionnalités du TP requièrent de l'aléatoire. Pour cela, veuillez utiliser les fonctions *rand* et *srand*. En particulier, soyez très prudent avec l'utilisation de *srand,* une mauvaise utilisation provoquera un résultat déterministe !

Travail à réaliser

Le travail présent a pour but d'ajouter de nouvelles fonctionnalités au jeu Polyland. Les mécaniques que nous allons ajouter sont inspirées d'un célèbre jeu, lui aussi basé sur des combats entre des créatures.

L'ajout de ces fonctionnalités permet d'introduire deux notions fondamentales de la programmation orientée objet : *les fonctions et classes génériques ainsi que la librairie STL*. La librairie STL est comme une boite à outil qui est à la disposition d'un programmeur. Elle lui permet d'éviter à réinventer la roue et elle lui simplifie la vie. Il faut juste apprendre à utiliser ses outils, afin de les maitriser et de s'en servir davantage.

Suite à une réunion avec les responsables design et gameplay, il vous est demandé de restructurer les mondes magiques. Jusqu'à lors, le seul monde magique que vous connaissiez était Polyland. Cependant, les responsables du gameplay veulent que le concept d'un monde magique devienne plus général qu'à l'application direct de Polyland.

Veuillez porter une attention particulière aux points suivants :

- **Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP4.**
- **Sauf mention explicite du contraire, c'est à vous de déterminer la visibilité de vos attributs (protected, private, public).**
- **L'un des principes du polymorphisme étant de limiter la duplication du code, pensez à utiliser au maximum les méthodes des classes mères.**
- **Lors de l'utilisation d'itérateurs, pensez à mettre à profit le mot clé auto!**
- **ATTENTION : Tout au long du TP, faites attention de toujours vérifier que vos itérateurs ne sont pas nuls et que vos conteneurs ne sont pas vides.**
- **ATTENTION : L'utilisation de boucles for ou while de la forme for(int i; i <vec.size(); i++) est interdit pour les nouvelles méthodes que vous devez implémenter. Vous devez soit utiliser les algorithmes lorsque possible, soit utiliser les boucles for/while en utilisant les itérateurs.**
- **ATTENTION : Afin d'éviter de créer des fichiers .cpp et .h en grande quantité, tous les foncteurs de ce TP doivent être implémentés dans un même fichier nommé Foncteur.h.**

Partie 1 :

Classe *MondeMagique*

La classe MondeMagique est une classe générique permettant de contenir des objets de type quelconque (qu'on appellera ici type T et type S).

- Cette classe devrait être implémentée seulement avec des algorithmes ou des méthodes du conteneur, aucune boucle n'est acceptée.
- Les instances T représentent les maîtres et les instances S représentent les compagnons. Par exemple, un monde magique pourrait contenir des sorciers (T) et des dragons (S) ou bien des fées (T) et des libellules (S) ou encore, comme dans Polyland, des dresseurs (T) et des créatures (S).

Les attributs suivants doivent être créés :

- Une *list* de pointeurs d'instances T listMaitre_ (utiliser le conteneur de la librairie STL).
- Une *list* de pointeurs d'instances S listCompagnon_ (utiliser le conteneur de la librairie STL).

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut.
- Un destructeur
- Des accesseurs pour chacun des conteneurs T et S du MondeMagique.
- Une méthode *ajouter* avec un paramètre de type T. La méthode ajoute le paramètre à la listT du MondeMagique. La méthode *ajouter* pour une instance S doit aussi être implémentée. Les deux instances (T et S) ont une relation d'agrégation avec leur list.
- Une méthode *supprimer* avec une instance de type T en paramètre. Cette méthode supprime un élément de la listT si ce dernier est == à celui passé en paramètre. Vous devez utiliser la fonction *find_if* de la STL et la méthode *erase* du conteneur. La méthode *supprimer* pour le type S doit aussi être implémentée.
- Une méthode *supprimerMaitre* qui prend un prédicat unaire en paramètre. Ce prédicat est un template de la méthode. Cette méthode supprime tous les éléments de la liste qui ne respecte pas ce prédicat. La méthode *supprimerCompagnon* doit aussi être implémentée.
- Une méthode vider qui vide les deux listes d'un coup.
- Les surcharges des opérateurs += et -= pour chaque type T et S qui ajoutent ou retirent une instance donnée dans la liste correspondante.

Partie 2 :

Classe *Polyland*

La classe *Polyland* hérite de la classe MondeMagique**.**

Les vecteurs de dresseurs et de créatures n'existent plus. Les seules méthodes qui ont été conservées du TP précédent sont : attrapperCreature et relacherCreature.

Elle implémente les méthodes suivantes :

- Une surcharge de l'opérateur << qui affiche d'abord la liste de dresseurs en ordre alphabétique de nom, puis la liste de créatures en ordre croissant d'attaque. Pensez à utiliser la fonction *sort* de la libraire STL.

Classe Dresseur

L'attribut suivant doit être modifié/ :

- Le vecteur de créature devient une liste de créatures.

Il est possible que l'utilisation d'une liste vous force à mettre à jour quelques fonctions (arguments, méthodes du conteneur utilisé).

La méthode suivante doit être modifiée :

- obtenirUneCreature, utilisez *find_if* de la STL avec un foncteur approprié pour tester la présence de la créature. Si la créature est absente renvoyez *nullptr.*
- operator==, doit utiliser find_if et un foncteur pour déterminer si les dresseurs ont les mêmes créatures (elles peuvent être dans un ordre différent).

Toutes les méthodes suivantes prennent un prédicat un argument. Il sera unaire ou binaire selon les cas. Ces méthodes sont des méthodes template, c'est-à-dire que leur argument est un template.

Les méthodes templates suivantes sont à ajouter :

- appliquerFoncteurUnaire, applique le foncteur à toutes les créatures du conteneur, notez que l'application du foncteur sur un élément peut modifier le foncteur.
- supprimerElement, supprime tous les éléments du conteneur pour lesquels l'application du foncteur passé en argument retourne *true*. Pensez à utiliser la méthode *remove_if* appropriée (celle de la STL? Celle du conteneur ?)
- obtenirCreatureMax, prend un foncteur en argument, et retourne la créature max selon l'ordre induit par le foncteur.

Classe Creature

La méthode template suivante est à ajouter :

- *trierPouvoirs*, prend un prédicat binaire en argument, et trie le vecteur de pouvoirs selon la relation d'ordre induite par le prédicat.

Classe FoncteurComparerCreatures

Ce foncteur prend en argument deux pointeurs de créature et renvoie *true* si la première créature a une attaque inférieure à celle de la deuxième. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Classe FoncteurComparerDresseurs

Ce foncteur prend en argument deux pointeurs de dresseurs et renvoie *true* si le premier a un nom strictement inférieur au nom du second. Cette comparaison se fera selon l'ordre alphanumérique. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Classe FoncteurComparerPouvoirs

Ce foncteur prend en argument deux pointeurs de pouvoir et renvoie *true* si le premier pouvoir a un nombre de dégâts inférieur à celui du deuxième. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Classe FoncteurCreaturesDeMemeNom

Ce foncteur prend en argument un pointeur de créature et renvoie *true* si cette créature a le même nom que l'attribut de la classe correspondant. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède l'attribut suivant :

- Une chaine de caractères qui désigne le nom de référence auquel sera comparé le nom de la créature passé en argument.

Il possède les méthodes suivantes :

- Un constructeur par paramètres qui initialise l'attribut nom

Classe FoncteurEgalCreatures

Ce foncteur prend en argument un pointeur de créature et renvoie *true* si cette créatures est égale à l'attribut creature du foncteur (au sens de leur opérateur ==). **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède l'attribut suivant :

- Un pointeur sur une créature.

Il possède la méthode suivante :

- Un opérateur() qui prend un pointeur sur une créature en argument et retourne vrai si les créatures (attribut + argument) sont égales.

Classe FoncteurCreatureVie

Ce foncteur prend en argument un pointeur vers une créature et incrémente une variable interne si les points de vie de la créature sont entre ces bornes. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède les attributs suivants :

- Un entier qui désigne la vie minimale
- Un entier qui désigne la vie maximale
- Un entier qui sert de compteur

Il possède les méthodes suivantes :

- Un constructeur par paramètres qui initialise les attributs qui désignent la vie minimale et la vie maximale. (Le compteur est initialisé à 0)
- Un opérateur() qui prend un pointeur sur une créature en argument
- La méthode d'accès à la valeur du compteur

Classe FoncteurGenerateurNombresAlea

Ce foncteur permet de générer un nombre aléatoire compris dans un intervalle. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède les attributs suivants :

- Un entier qui désigne la borne inférieure de l'intervalle
- Un entier qui désigne la borne supérieure de l'intervalle

Il possède les méthodes suivantes :

- Un constructeur par paramètres qui initialise les bornes de l'intervalle
- Un constructeur par défaut qui initialise l'intervalle à [0,6]
- Un opérateur() qui retourne un nombre aléatoire

Classe AttaqueMagiquePoison

Il possède l'attribut suivant :

- Un FoncteurGenerateurNombresAlea

La méthode suivante est à modifier :

- *appliquerAttaque* doit utiliser le foncteur pour les tirages aléatoires
- Le constructeur doit initialiser le nouvel attribut pour que le tirage soit fait dans l'intervalle [1,6].

Classe AttaqueMagiqueConfusion

Il possède l'attribut suivant :

- Un FoncteurGenerateurNombresAlea

La méthode suivante est à modifier :

- *appliquerAttaque* doit utiliser le foncteur pour les tirages aléatoires
- Le constructeur doit initialiser le nouvel attribut pour que le tirage soit fait dans l'intervalle [1,6]..

Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées.

La fin du *main* comporte une section à compléter qui traite des *maps*. **N'oubliez pas de compléter ses dernières lignes de code.**

Le résultat final devrait être similaire à ce qui suit :

```
TEST AFFICHAGE
pouvoir de la creature avant trie
Pokachu a 10 en attaque et 20 en defense,
Il a 100/100 PV et 50/50 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Tonnerre possede un nombre de dégat de 15 et une energie necessaire de 10

Eclair possede un nombre de dégat de 10 et une energie necessaire de 5



pouvoir de la creature apres trie
Pokachu a 10 en attaque et 20 en defense,
Il a 100/100 PV et 50/50 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Eclair possede un nombre de dégat de 10 et une energie necessaire de 5

Tonnerre possede un nombre de dégat de 15 et une energie necessaire de 10



Affichage de Polyland
Sacha possede 2 creature(s) et appartient a l'equipe UDEM

Vous possede 1 creature(s) et appartient a l'equipe PolyMtl

Pokachu a 10 en attaque et 20 en defense,
Il a 100/100 PV et 50/50 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Eclair possede un nombre de dégat de 10 et une energie necessaire de 5

Tonnerre possede un nombre de dégat de 15 et une energie necessaire de 10
```

```
Pokachoum a 10 en attaque et 7 en defense,
Il a 100/100 PV et 50/50 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Pokachoum ne connait aucun pouvoir

Touflamme a 15 en attaque et 15 en defense,
Il a 110/110 PV et 50/50 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Tonnerre possede un nombre de dégat de 15 et une energie necessaire de 10



Salimouche a 20 en attaque et 15 en defense,
Il a 110/110 PV et 50/50 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Boule de feu possede un nombre de dégat de 5 et une energie necessaire de 5




FIN TEST AFFICHAGE

TEST DRESSEUR
appliquerFoncteurUnaire: OK
TEST DRESSEUR : obtenir element max
Salimouche a 20 en attaque et 15 en defense,
Il a 110/110 PV et 50/50 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Boule de feu possede un nombre de dégat de 5 et une energie necessaire de 5



TEST DRESSEUR : FIN obtenir element max
TEST DRESSEUR : suppression
```

Correction

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

- (10 points) Compilation et exécution exacte du programme.
- (3 points) Utilisation exacte des conteneurs et des boucles.
- (3 points) Utilisation adéquate des algorithmes et des foncteurs.
- (2 points) Utilisation adéquate de la classe générique et du foncteur générique.
- (2 points) Documentation du code et norme de codage.