INF1010

Programmation Orientée-Objet

Travail pratique #2
Vecteurs et surcharge d'opérateurs

Objectifs :	Permettre à l'étudiant de se familiariser avec la surcharge d'opérateurs, les vecteurs de la librairie STL et l'utilisation du pointeur *this*.
Remise du travail :	Mardi 4 Octobre 2016, 8h
Références :	Notes de cours sur Moodle & Chapitre 14 du livre Big C++ 2e éd.
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 (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

Travail à réaliser

Le travail consiste à continuer le programme du jeu Polyland commencé au TP précédent en y intégrant les notions de vecteurs et de surcharge d'opérateurs.

Pour remplacer les tableaux dynamiques qui limitaient le nombre de dresseurs, de créatures et de pouvoirs, ce TP fait appel aux vecteurs de la librairie STL, soit std::vector. Et, pour faciliter les interactions avec les différents objets, la surcharge d'opérateurs sera utilisée.

Les vecteurs implémentés en C++ (STL) sont très pratiques : ce sont des tableaux dont la taille est dynamique. On peut y ajouter des éléments sans se préoccuper de la taille de notre vecteur étant donné que la gestion de la mémoire est automatique.

Le langage C++ est un langage avec lequel il est possible de redéfinir la manière dont fonctionnent la plupart des opérateurs (arithmétiques (+, -, *, /), d'affectation, etc..) pour des nouvelles classes. Nous pouvons donc redéfinir le comportement de ces opérateurs afin qu'ils effectuent une nouvelle opération ou englobent plusieurs opérations pour ces nouvelles classes.

Pour vous aider, les fichiers du TP précédent vous sont fournis. Vous n'avez qu'à implémenter les nouvelles méthodes décrites plus bas. Les attributs qui ne sont plus nécessaires ont été supprimés. Et les méthodes à modifier vous ont été indiquées.

ATTENTION : Tout au long du TP, assurez-vous d'utiliser les opérateurs sur les objets et non sur leurs pointeurs ! Vous devez donc déréférencer les pointeurs si nécessaires.

ATTENTION : Vous serez pénalisés pour les utilisations inutiles du mot-clé *this*. Utilisez-le seulement où nécessaire.

ATTENTION : Il est fortement recommandé d'utiliser les fichiers fournis, plutôt que de continuer avec vos fichiers du TP1.

Classe *Dresseur*

Créer une classe *Dresseur* qui représente un joueur du monde de *PolyLand*.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Cette classe contient les attributs supplémentaires suivants :

- Un Vecteur de pointeurs de *Creature* (remplace le tableau)
- Une équipe (string)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les quatre attributs aux valeurs par défaut.
- Un constructeur qui prend un paramètre un nom et une équipe et qui initialise les attributs correspondants. Les autres attributs seront initialisés par défaut.
- Les méthodes d'accès et de modification des attributs.
- Un destructeur (doit être modifié si requis).
- La méthode *obtenirUneCreature* qui prend en paramètre le nom d'une créature et qui retourne cette créature. **Pensez à utiliser l'opérateur == surchargé pour une *Créature.***
- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernent un *Dresseur*, tel que présenté dans l'exemple à la fin du document.
- L'opérateur == qui prend un *Dresseur* en paramètre et permet de comparer 2 dresseurs en considérant seulement le vecteur de créatures qu'il possède (le vecteur doit contenir les mêmes créatures et nécessairement le même nombre de créature). L'opérateur retourne *true* si les dresseurs ont toutes les mêmes créatures, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *dresseur1 == dresseur2*. **Pensez à utiliser l'opérateur == surchargé pour une *Créature.*** Attention : les créatures ne sont pas nécessairement dans le même ordre.
- L'opérateur == qui prend un nom en paramètre et permet de comparer 2 dresseurs en considérant seulement le nom. L'opérateur retourne *true* si les noms sont identiques, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *dresseur == nom*.
- L'opérateur == de type *friend* qui permet d'inverser l'ordre de l'opérateur== précédent. Ainsi, cet opérateur va pouvoir faire l'opération *nom == dresseur*.

Classe *Creature*

Créer une classe *Creature* qui représente une créature dans le monde de Polyland.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Cette classe possède les attributs supplémentaires suivants :

- Un Vecteur de pointeurs de *Pouvoir* (remplace le tableau)

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut qui initialise les attributs aux valeurs par défaut.

- Un constructeur par paramètres qui initialise les attributs nom, attaque, defense, point de vie, et énergie selon les paramètres correspondants. Les autres paramètres sont initialisés à des valeurs par défaut.
- Un constructeur par copie qui va copier tous les attributs.
- Un opérateur = qui écrase les attributs de l'objet de gauche par les attributs l'objet passés en paramètre.
- Un destructeur (doit être modifié si requis).
- Les méthodes d'accès et de modification des attributs.
- Une méthode d'ajout et une de retrait d'un pouvoir (*apprendrePouvoir* et *oublierPouvoir*). Pour le retrait, basez-vous uniquement sur le nom du pouvoir et **pensez à utiliser l'opérateur == surchargé pour un *Pouvoir***.
- L'opérateur == qui prend une *Creature* en paramètre et permet de comparer 2 créatures en considérant tous les attributs sauf le vecteur de *Pouvoir* et le nombre de pouvoir. L'opérateur retourne *true* si les créatures ont les mêmes attributs, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *creature1 == creature2*.
- L'opérateur == qui prend un nom en paramètre et permet de comparer 2 créatures en considérant seulement le nom. L'opérateur retourne *true* si les noms sont identiques, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *créature == nom*.
- L'opérateur == de type *friend* qui permet d'inverser l'ordre de l'opérateur== précédent. Ainsi, cet opérateur va pouvoir faire l'opération *nom == créature*.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernant une *Creature*, tel que présenté dans l'exemple à la fin du document. Pensez à appeler l'opérateur << de la classe *Pouvoir*.

Classe *Pouvoir*

La classe *Pouvoir* sert à représenter les pouvoirs d'une créature du monde de Polyland.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Les méthodes suivantes doivent être implémentées :

- Un constructeur par copie qui va copier tous les attributs.
- Un opérateur = qui écrase les attributs de l'objet de gauche par les attributs l'objet passés en paramètre.
- L'opérateur == qui prend un *Pouvoir* en paramètre et permet de comparer 2 pouvoirs en considérant seulement le nom. L'opérateur retourne *true* si les pouvoirs ont le même nom, *false* sinon. Ainsi, cet opérateur va pouvoir faire l'opération *pouvoir1 == pouvoir2*.
- L'opérateur <<, qui affiche les informations qui concernant un *Pouvoir*, tel que présenté dans l'exemple à la fin du document.

Classe ObjetMagique

La classe *ObjetMagique* sert à représenter les objets magiques utilisables sur une créature du monde de Polyland.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Les méthodes suivantes doivent être implémentées :

- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernant un *ObjetMagique*, tel que présenté dans l'exemple à la fin du document.

Classe *Polyland*

La classe *Polyland* est celle qui fait le lien entre toutes les classes précédentes.

Les attributs et méthodes liées au TP1 restent inchangées, sauf si spécifié.

Elle contient les attributs supplémentaires suivants :

- Vecteur de pointeurs de *Dresseurs* (remplace le tableau)
- Vecteur de pointeurs de *Creatures* (remplace le tableau)

Elle implémente les méthodes suivantes :

- Un constructeur par défaut.
- Un destructeur (doit être modifié si requis).
- La méthode *ajouterDresseur* qui permet d'ajouter un dresseur reçu en paramètre ; deux dresseurs ne peuvent pas avoir le même nom. **Pensez à utiliser l'opérateur == surchargé pour un *Dresseur.***
- La méthode *retirerDresseur* qui permet de retirer le dresseur en utilisant le nom reçu en paramètre. **Pensez à utiliser l'opérateur == surchargé pour un *Dresseur.***
- L'opérateur += qui prend en paramètre un dresseur, qui l'ajoute au vecteur de dresseurs et qui retourne la Polyland avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur être très similaire à *ajouterDresseur*. Évitez de recopier du code.
- L'opérateur -= qui prend en paramètre un dresseur, qui l'enlève du vecteur de dresseurs et qui retourne la Polyland avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur être très similaire à *retirerDresseur*. Évitez de recopier du code.
- La méthode *ajouterCreature* qui permet d'ajouter une créature reçue en paramètre ; deux créatures ne peuvent pas avoir le même nom. **Pensez à utiliser l'opérateur == surchargé pour une *Creature.***
- La méthode *retirerCreature* qui permet de retirer la créature en utilisant le nom reçu en paramètre. **Pensez à utiliser l'opérateur == surchargé pour une *Creature.***
- L'opérateur += qui prend en paramètre une créature, qui l'ajoute au vecteur de créatures et qui retourne la Polyland avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur être très similaire à *ajouterCreature*. Évitez de recopier du code.
- L'opérateur -= qui prend en paramètre une créature, qui l'enlève du vecteur de créatures et qui retourne la Polyland avec les nouvelles modifications. Le comportement de cette surcharge d'opérateur être très similaire à *retirerCreature*. Évitez de recopier du code.
- L'opérateur << (remplace la méthode d'affichage), qui affiche les informations qui concernant *Polyland*, tel que présenté dans l'exemple à la fin du document.

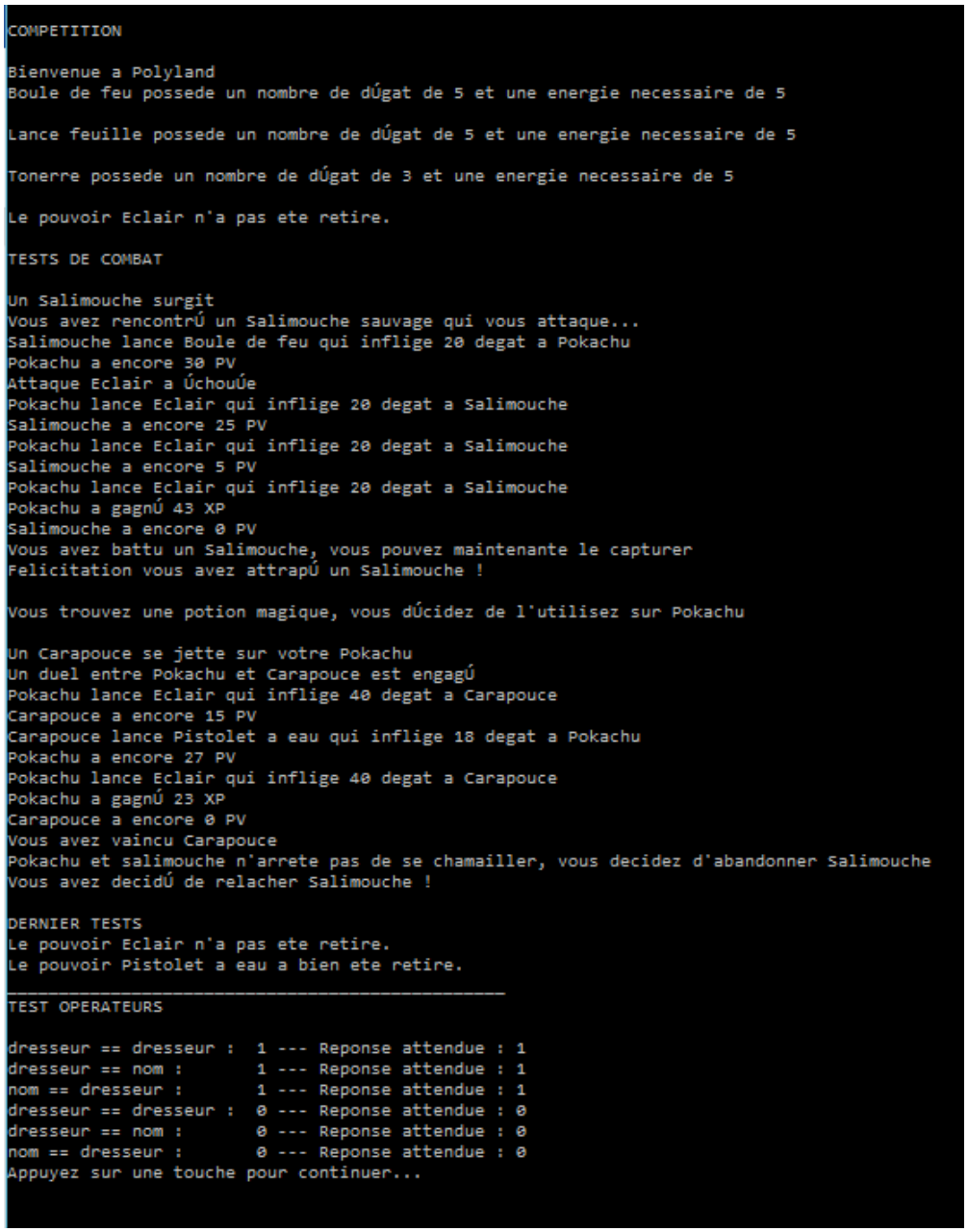
Aide : Pensez à utiliser les différentes méthodes pour afficher les informations des différentes classes.

Main.cpp

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

Le résultat final devrait être similaire à ce qui suit :

```
CREATION DES DRESSEURS

CREATION ET AFFICHAGE DES CREATURES
Pokachu a 10 en attaque et 2 en defense,
Il a 50/50 PV et 25/25 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 100 jusqu'au prochain niveau
Pouvoirs :
Pokachu ne connait aucun pouvoir

Touflamme a 15 en attaque et 3 en defense,
Il a 45/0 PV et 20/0 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 0 jusqu'au prochain niveau
Pouvoirs :
Touflamme ne connait aucun pouvoir

Pokachoum a 10 en attaque et 7 en defense,
Il a 50/0 PV et 25/0 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 0 jusqu'au prochain niveau
Pouvoirs :
Pokachoum ne connait aucun pouvoir


AJOUT DE CREATURES ET DE DRESSEURS A POLYLAND

Salimouche a bien Útú ajoutú !
Carapouce a bien Útú ajoutú !
Balbazar a bien Útú ajoutú !
Pokachu a bien Útú ajoutú !
Touflamme a bien Útú ajoutú !
Pokachoum a bien Útú ajoutú !
Regis a bien Útú ajoutú !
Pierre a bien Útú ajoutú !
Sasha a bien Útú ajoutú !
Pierre n'a pas ete ajoute
TEST D'AFFICHAGE

Regis possede 0 creature(s) et appartient a l'equipe Equipe de Poly
Pierre possede 0 creature(s) et appartient a l'equipe Equipe de Poly
Sasha possede 0 creature(s) et appartient a l'equipe Team de feu

Pokachoum a 10 en attaque et 7 en defense,
Il a 50/0 PV et 25/0 Energie
Il est au niveau 1 avec 0d'XP
Il lui manque 0 jusqu'au prochain niveau
Pouvoirs :
Pokachoum ne connait aucun pouvoir

Pierre possede 0 creature(s) et appartient a l'equipe Equipe de Poly
```



Question

1. Quelle est l'utilité de l'opérateur = et du constructeur par copie ?
2. Dans quel cas est-il absolument nécessaire de les implémenter ?
3. Qu'est-ce qui différencie l'opérateur = du constructeur par copie ?

Correction

La correction du TP1 se fera sur 20 points.

Voici les détails de la correction :

- (3 points) Compilation du programme ;
- (3 points) Exécution du programme ;
- (4 points) Comportement exact des méthodes du programme ;
- (3 points) Surcharge correcte des opérateurs ;
- (2 points) Utilisation correcte des vecteurs ;
- (1.5 points) Documentation du code ;
- (1 point) Utilisation correcte du mot-clé *this* pour les opérateurs ;
- (1 point) Utilisation correcte du mot-clé *const* ;
- (1.5 points) Réponse à la question.