[Question P1.1]

Comment représentez-vous ces vecteurs ?

Chaque vecteur sera représenté comme un tableau dynamique (“vector”) de double du au fait que l’on travaille avec des vecteurs de dimension inconnue au préalable et dont les valeurs ne seront sûrement pas que des entiers. De plus, on a choisi de représenter nos vecteurs avec un “vector” et pas avec un “array” puisque l’énoncé nous demande de faire une fonction “augmente” qui changerait la taille du tableau. Or ceci n’est pas possible avec des tableau de type array.

Comment sont-ils organisés : quels attributs ?

Comme seul attribut on a le tableau dynamique décrit dans la question précédente.

Quelles méthodes ?

Dans notre classe Vecteur on a 15 méthodes, les 12 demandées et 3 méthodes en plus:

1. Une méthode qui vérifie si deux vecteurs sont de même dimension: “essayer_dim” On a créé cette méthode pour éviter la duplication de code car dans plusieurs méthodes (produit scalaire, addition, soustraction) on a besoin de faire cette vérification.
2. Une méthode pour retourner la i-ème composante du vecteur en question: “getComp”. Pour l’instant on n’a pas besoin de cette méthode mais on a cru que celle-ci pourrait être utile plus tard dans notre projet.
3. Une méthode “size” qui retourne la taille du tableau dynamique qui a la classe Vecteur comme attribut.

Quels droits d'accès ?

Pour l’instant on a structuré notre classe de la façon suivante. C’est possible qu’au fur et à mesure que le projet avance nous devons changer quelques droits d'accès.

Privé:

1. Le tableau où l’on stocke les coordonnées du vecteur;
2. La méthode “essayer_dim” décrite précédemment;
3. La méthode “norme2” qui calcule le carré de la norme du vecteur et qui pour nous s’agit que d’une méthode auxiliaire pour calculer la norme d’un vecteur;
4. La méthode “oppose” qui calcule l’opposé d’un vecteur et qui pour nous, à ce stade là s’agit que d’une méthode auxiliaire pour effectuer une soustraction de vecteurs en utilisant la méthode d’addition.

Publique:

1. Les méthodes qui effectue des opérations: addition, soustraction, produit scalaire, produit vectoriel, multiplication par un scalaire et norme. Ces méthodes doivent être publiques si on veut les appelées depuis le main pour faire les tests.
2. Les méthodes “augmente”, “set_coord”, “getComp” et “size”.
3. La méthode pour afficher un vecteur.

[Question P1.2]

Quel choix avez vous fait pour les opérations entre vecteurs de dimensions différentes?

Nous avons choisi de lancer une exception parce que nous ne considérons pas très rigoureux les autre choix du point de vue mathématique. Ces exceptions seront attrapés (“catch”) dans le main. C’est-à-dire, si jamais on se trompe et on essaye d’effectuer une opération entre vecteurs de dimensions différentes, le programme s'arrêtera et affichera sur l’écran d’où provient l’erreur.

[Question P4.1]

Avez-vous ajouté un constructeur de copie ? Pourquoi (justifiez votre choix) ?

Non, nous considérons que c’est n’est pas nécessaire de redéfinir le constructeur de copie. Vu que nous n’utilisons pas de pointeurs le constructeur de copie par défaut nous suffit.

[Question P4.2] Si l'on souhaitait ajouter un constructeur par coordonnées sphériques (deux angles et une longueur) pour les vecteurs de dimension 3,

A. Que cela impliquerait-il au niveau des attributs de la classe ?

Au niveau des attributs ça ne change rien, l'idée c'est de faire en sorte que l'interface propose les coordonnées sphérique mais rien ne nous oblige d'enregistrer les données en coordonnées sphérique. On peut très bien juste faire la conversion lors de l'appel au constructeur.

B. Quelle serait la difficulté majeure (voire l'impossibilité) de sa réalisation en C++ ? (C'est d'ailleurs pour cela qu'on ne vous demande pas de faire un tel constructeur !)

La difficulté c'est qu’alors le type et le nombre des paramètres sont potentiellement les même que pour un constructeur d'un vecteur 3D en coordonnées cartésienne. Il n'y a dans ce cas pas la possibilité de surcharger le constructeur 3 double et donc d'offrire les deux possibilités. Une solution pourrait par exemple être d'ajouter un paramètre supplémentaire comme un bool qui indique le format des arguments (cartésien ou sphérique).

[Question P4.3]

Quels opérateurs avez vous introduits ?

Nous avons introduit les opérateurs suivants:

⇒ operator== et operator!= qui font référence à la méthode “compare”

⇒ operator+=

⇒ operator+ qui fait référence à la méthode “addition”

⇒ operator-=

⇒ operator- qui fait référence à la méthode “soustraction”

⇒ operator<< pour l'affichage des vecteurs, fait référence à la méthode “affiche”

⇒ operator* qui en écrivant vecteur*a ou a*vecteur calcule le résultat de la multiplication par un scalaire a pour un vecteur (nous avons fait une surcharge externe pour a*v et une surcharge interne pour v*a)

⇒ operator* qui fait référence à la méthode relative au produit scalaire

⇒ operator^ qui fait référence à la méthode relative au produit vectoriel des vecteurs de dimension 3

[Question P6.1]

Comment avez vous conçu votre classe Intégrateur ?

Pour nous la classe Intégrateur est une classe abstraite qui aura un seul attribut de type double associé au pas de temps et une fonction virtuelle pure “integre” qui prendra un oscillateur et un temps(double) comme paramètre et qui fera évoluer notre oscillateur de ce temps spécifié.

[Question P6.2]

Quelle est la relation entre les classes Intégrateur et IntegrateurEulerCromer ?

Un IntegrateurEulerCromer **est un** Intégrateur, donc la classe IntegrateurEulerCromer est une sous classe de la classe Intégrateur.

[Question P7.1]

Comment se situent ces classes par rapport à la classe Oscillateur définie la semaine passée ?

Un pendule **est un** oscillateur et un ressort **est** aussi **un** oscillateur.

Par conséquent, la classe Pendule et la classe Ressort sont des sous classe de la classe Oscillateur..

Nous avons décidé de transformer la classe Oscillateur en une classe abstraite puisque nous ne pouvons pas définir une équation du mouvement général pour tous les types d’oscillateur; en effet, l’équation du mouvement est propre à chaque type d’oscillateur et doit être spécifiée dans toutes les sous classe d’oscillateur. (i.e. la méthode “eq_mvt” est maintenant une méthode virtuelle pure)

[Question P8.1]

En termes de POO, quelle est donc la nature de la méthode dessine() ?

La méthode dessine est une méthode de caractère polymorphique de type void.

[Question P8.2]

Quelle est la bonne façon de le faire dans un cadre de programmation orientée-objet ?

La classe Système doit avoir une collection hétérogène d’oscillateurs. Pour ceci, on doit utiliser des pointeurs. De la même façon, on a choisi que la classe système ait aussi comme attribut un pointeur sur un Intégrateur. Pour notre programme on a choisit d’utiliser des pointeurs à la C.

Pour nous, un système a ses propres oscillateurs et donc sa destruction comporte la destruction de ses oscillateurs; cependant, nous considérons que plusieurs systèmes pourraient avoir le même intégrateur et donc ce n’est pas le système qui doit détruire l’intégrateur.

De plus, la classe Système a besoin d’une méthode qui fait évoluer le système de façon à que chaque oscillateur évolue selon leur propre équation du mouvement. Or, puisqu’on travaille avec des pointeurs il suffit d’effectuer une boucle for dans laquelle on appelle la fonction “integre” de l'Intégrateur choisi pour notre système en passant à chaque itération un objet de notre collection.

[Question P8.3]

A quoi faut-il faire attention pour les classes contenant des pointeurs ? Quelles solutions peut-on envisager ?

Il faut faire attention au constructeur de copie, à l'opérateur d'affectation et aussi au destructeur. Pour le constructeur de copie, soit on peut déclarer une méthode virtuel pure dans la classe mère qui soit redéfinie dans les sous classes pour faire une copie polymorphique, soit on peut l'effacer si nous considérons que ça n'a pas trop de sens de pouvoir faire une copie d'une instance de la classe. Dans les deux cas, il faut re-définir ou effacer l'opérateur d'affectation.
Pour le destructeur, il faut faire attention à si les objets pointés sont propres aux systèmes ou si sont partagés. Tel que décrit dans la question précédente,dans le destructeur de la classe système il faudrait détruire les oscillateurs qui le composent mais pas l'intégrateur.

[Question P8.4]

Comment représentez vous la classe Système ? Expliquez votre conception (attributs, interface, …)

Un système **est un** objet dessinable. Donc, la classe Système est une sous classe de la classe Dessinable.
Comme nous avons déjà expliqué, selon nous la classe Système aura deux attributs:

1. Une collection hétérogène d'oscillateurs: "vector<Oscillateur*> collection_osc"
2. Un pointeur sur un intégrateur: "Integrateur* integrateur"

Pour nous les oscillateurs seront propres au système et l'intégrateur, par contre, pourra être partagé.
En ce qui concerne la partie publique de notre classe Système, nous avons défini un constructeur qui prend comme paramètre un pointeur vers un SupportADessin et fait appel au constructeur de la classe Dessinable; nous avons défini un constructeur qui détruit les oscillateurs du systèmes; et nous avons implémenté 5 méthodes:

1. Une méthode "dessine" hérité de la classe Dessinable.
2. Une méthode "évolue" qui fait évoluer les oscillateurs qui font partie du système. Celle-ci appelle la fonction "integre" de l'intégrateur du système avec chacun des oscillateurs qui composent le système au moyen d'une boucle for.
3. Une méthode "affiche" qui affiche les différents types d'oscillateurs.
4. Une méthode "ajouterOscillateur" qui prend un pointeur vers un Oscillateur et l'ajoute au système.
5. Une méthode "ajouterIntegrateur" qui prend un pointeur vers un Intégrateur et indique au système quel intégrateur utiliser en affectant l'attribut de classe.

Rappel: Impossible de copier des systèmes et d'utiliser l'opérateur d'affectation.

[Question P11.1]

Comment représentez-vous ces nouveaux oscillateurs ? Où s'incrivent-ils dans votre conception ?

Nous avons décidé d'implémenter les oscillateus couplés Pendule Ressort et Pendule Double que l'on a nommé RessortPenduleCouplee et Pendule_Double respectivement.

Or, un oscillateur couplée **est un** oscillateur.

Les classe RessortPenduleCouplee et Pendule_Double sont des sous classes de la classe Oscillateur.

Nous les avons représentées en suivant la même structure que nous avons utilisé pour représenter un ressort ou un pendule.

Dans ce cas, comme attribut spécifique d'un ressort pendule/ pendule ressort on a une longueur au repos(double) et un raideur(double) et pour un pendule double on a deux longueur(double) et deux masses(double).

[Question P12.1]

Comment implémentez-vous la possibilité de tracer les trajectoires dans l'espace des phases ?

Dans notre projet, l'implémentation de la possibilité de tracer les trajectoires dans l'espace des phases se base sur le fait que chaque oscillateur possède un tableau comme attribut où l'on stocke à chaque pas de temps le point de sa trajectoire dans l'espace des phases. Cette implémentation est loin d'être optimale car nous occupons beaucoup de mémoire pour garder tous nos points. En effet, nous avons dérouler notre programme pour plus de 5 minutes et rien d'important c'est passé mais nous pouvons supposer que si nous le déroulons assez longtemps l'ordinateur atteindrait sa capacité maximale.

Pour une description plus exhaustive des changements que l'on a dû effectuer pour pouvoir tracer les trajectoires dans l'espace des phases, voir fichier CONCEPTION.

[Question P13.1]

Où cela s'intègre-t-il dans votre projet/conception ? Quels changements cela engendre ?

Nous avons décidé d'implémenter l'intégrateur de Newmark.

Pour ceci, on a dû ajouter une classe IntegrateurNewmark qui est une sous classe de la classe abstraite Intégrateur. Nous l'avons doté de la même structure que la classe IntegrateurEulerCramer.

Nous n'avons pas effectuer des changements ailleurs.