IN204
Programmation Orientée Objet – Examen de mise en œuvre des notions de C++

Examen du 17 novembre 2025

B. Monsuez

NOM :	
PRENOM :	

Question n°1: Constructeurs

Nous nous intéressons à la classe suivante qui définit un vecteur en 2 dimensions :

```
class vector2D
{
private:
    double x;
    double y;

public:
    vector2D()
    {...}
    vector2D(double X, double Y)
    {...}
    interval(const vector2D& anotherVector2D)
    {...}
};
```

Question n°1.1:

Pour chacun des constructeurs, expliquez la fonction du constructeur et indiquez si le constructeur est un constructeur que C++ génère automatique en son absence ou pas.

```
    vector2D()
```

```
vector2D(double X, double Y)
```

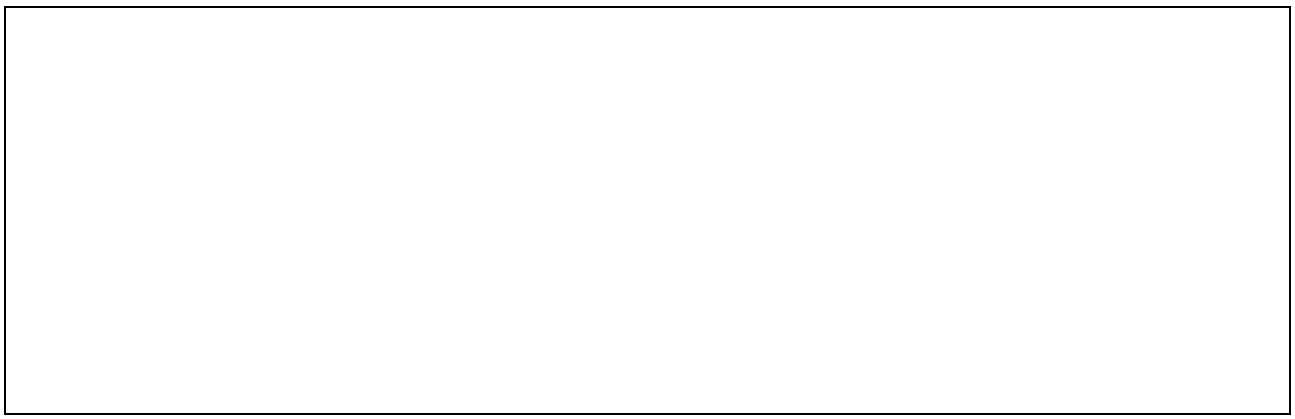
```
vector2D(const vector2D& anotherVector2D)
```

Question n°1.2:

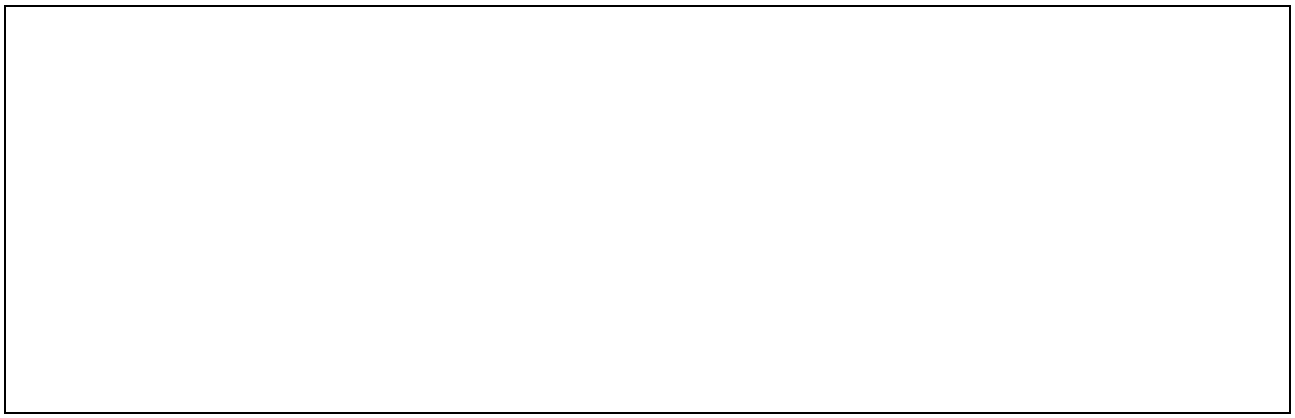
Écrivez pour chacun des constructeurs le code d'initialisation que le constructeur doit implanter.

```
vector2D()
```

```
vector2D(double X, double Y)
```



```
vector2D(const vector2D& anotherVector2D)
```



Question n° 2 : Accéder aux données internes stockées dans les champs

Question n° 2.1 : Champs privés

Est-il possible d'accéder aux champs x et y ? Pourquoi donc ?

Question n° 2.2 :

Proposez une méthode ou plusieurs méthodes pour accéder aux deux champs x et y en lecture seulement.

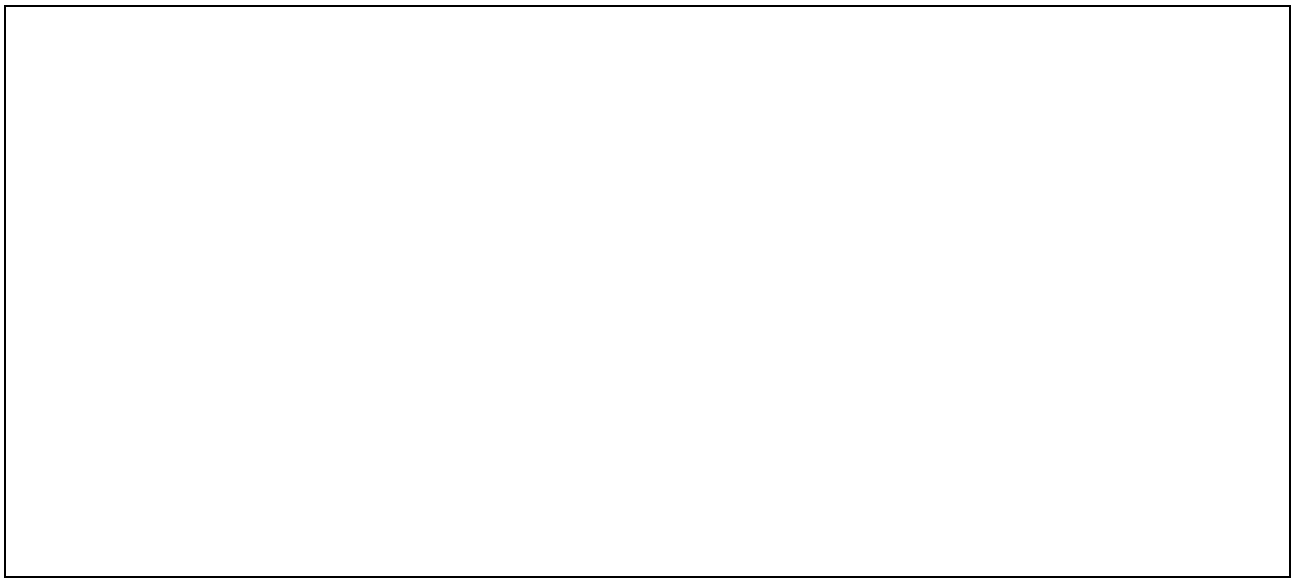
Question n° 3 : Représentation polaire

Le constructeur `vector2D(double X, double Y)` prend deux arguments, les coordonnées en X et en Y. Nous aimerions pouvoir créer un vecteur à partir de sa représentation polaire avec comme premier argument la norme du vecteur et comme second argument l'angle du vecteur.

Question n°3.1

Comment pourriez-vous pour implanter une telle fonction ? Proposer une solution.

> Pensez éventuellement à une méthode statique que l'on pourrait par exemple nommer `fromPolar`:
>
> ```
> static vector2D fromPolar (...) {}
> ```



Question n°3.2

Proposez un ensemble de méthodes permettant de récupérer à la fois :

- De récupérer la norme du vecteur intitulée « `norm()` »
- De récupérer l'angle du vecteur « `angle()` »,
- Ainsi qu'une paire composée de la norme et l'angle du vecteur « `toPolar()` ».

Question n°3.3

Répondez aux questions suivantes :

- Les dernières fonctions modifient-elles l'état de l'objet (ie. les champs internes de l'objet) ?
- Si ces fonctions ne modifient pas l'état de l'objet, comment indique-t-on au compilateur que ces fonctions ne doivent pas modifier l'état de l'objet ?
- Quelles sont les vérifications qu'effectue le compilateur quand il est indiquée qu'une fonction en modifie pas l'impact de l'objet ?

Question n° 4 : Opérateurs testant l'égalité et l'inégalité.

Nous allons définir les opérations de comparaison. Nous commençons d'abord par l'opérateur d'égalité :

```
bool operator == (const vector2D& anotherVector2D) const
```

Question n° 4.1

Expliquer la syntaxe de l'opérateur d'égalité puis donner son implantation.

Question n° 4.2

Nous nous intéressons à l'opération d'inégalité. Quand est-ce qu'il est nécessaire de définir une opération d'inégalité ? Et quand il est nécessaire de définir une opération d'inégalité, donnez le code de cette opération d'inégalité

Question n° 5 : Un peu de calcul

Nous nous proposons de définir les opérations habituelles sur les vecteurs. Nous débutons par les opérations d'additions (+) at de soustractions.

Question n° 5.1 :

Proposez une implantation des deux opérateurs qui prend deux vecteurs et retourne un nouvel vecteur étant l'addition, (resp.la soustraction) des deux vecteurs arguments.

Question n° 5.2 :

Proposez une implantation des deux opérateurs qui prend deux vecteurs et additionne (resp. soustrait) au premier vecteur le second vecteur.

Pensez aux opérateurs « += », …

Question n° 5.3 :

On se propose d'implanter le produit scalaire deux vecteurs en utilisant l'opérateur « . », proposez une implantation qui implante cette opération.

Question n° 6: Patrons

La classe `vector2D` est définie pour des vecteurs de type de type `double`. En fait, les vecteurs peuvent être définis pour tous les nombres flottants et plus généralement.

Question n° 6.1

Proposer une transformation de la classe `interval` pour la paramétrer par un type représentant un nombre.

Question n° 6.2

En C++ 20, il est possible d'ajouter des contraintes sur les paramètres d'une fonction ou d'une classe. Déterminer les opérateurs ou les méthodes que le paramètre de la classe intervalle doit fournir pour que le code compile correctement.

Ensuite, proposer un ajout de contrainte pour vérifier dès la déclaration que le type paramètre fournit bien les opérateurs et méthodes attendues.

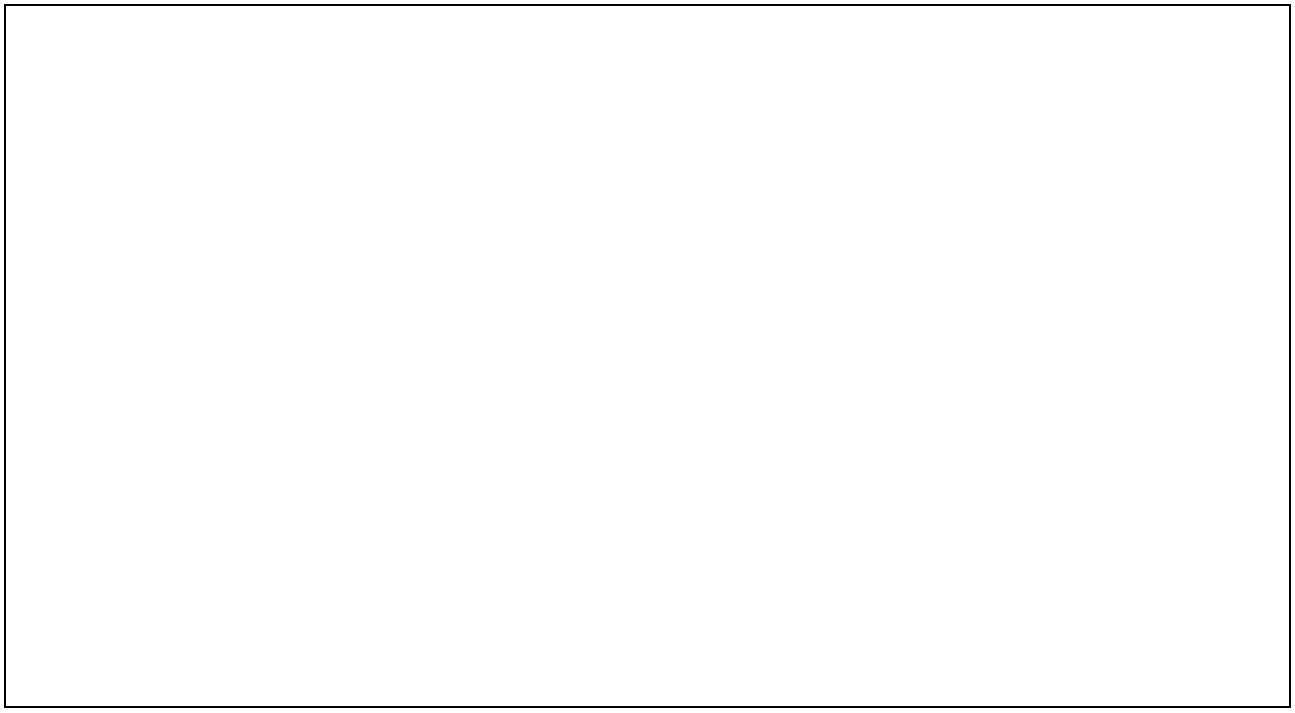
Nous rappelons que la bibliothèque standard de C++ fournit les concepts suivants :

```
template< class T >
concept integral; // Indique que le type T est un type entier (signé ou non s
igné).

template< class T >
concept floating_point; // Indique que le type T est un type de nombre à virg
ule flottant.

template< class T >
concept equality_comparable; // Indique que le type T supporte les opérateurs
== et !=.

template< class T >
concept totally_ordered; // Indique que le type T supporte les opérateurs ==
et !=, <, <=, >, >=
```



Question n° 7: Opérateurs surchargés & flux

Nous souhaitons écrire sur un flux de type. Pour ce faire, nous envisageons de définir un opérateur << qui a la signature suivante :

```
template<type T, typename charT, typename traits>
std::basic_ostream<charT, traits>& operator << (std::basic_ostream<charT, tra
its>& aStream, const vector2D<T>& theInterval);
```

Question n°7.1 :

Pourquoi cet opérateur ne peut pas être défini comme opérateur dans la classe `vector2D<T>` ?

Question n°7.2 :

Proposer une implantation qui pour l'objet `vector2D<float>(0.0, 1.5)` affiche (0.0, 1.5).

Question n° 8: Dérivation

Nous souhaitons définir une classe dérivée de `vector2D` qui représente un vecteur à 3 dimensions que nous appellerons `vector3D`. Nous souhaitons que cette classe `vector3D` hérite de la classe `vector2D.`

Question n° 8.1:

Construisez une classe :

```
template<typename T> class vector3D ...
```

qui devra hériter de la classe `vector2D` et qui devra fournir les constructeurs suivants:

```
public:
    vector3D() {…}
    vector3D(T X, T Y, T Z) {…}
    vector3D(const vector3D<T>& anotherVector);
```

Question n° 8.2:

Expliquer pourquoi les champs x et y définis dans le vecteur vector2D ne sont pas accessibles dans le vecteur `vector3D`.

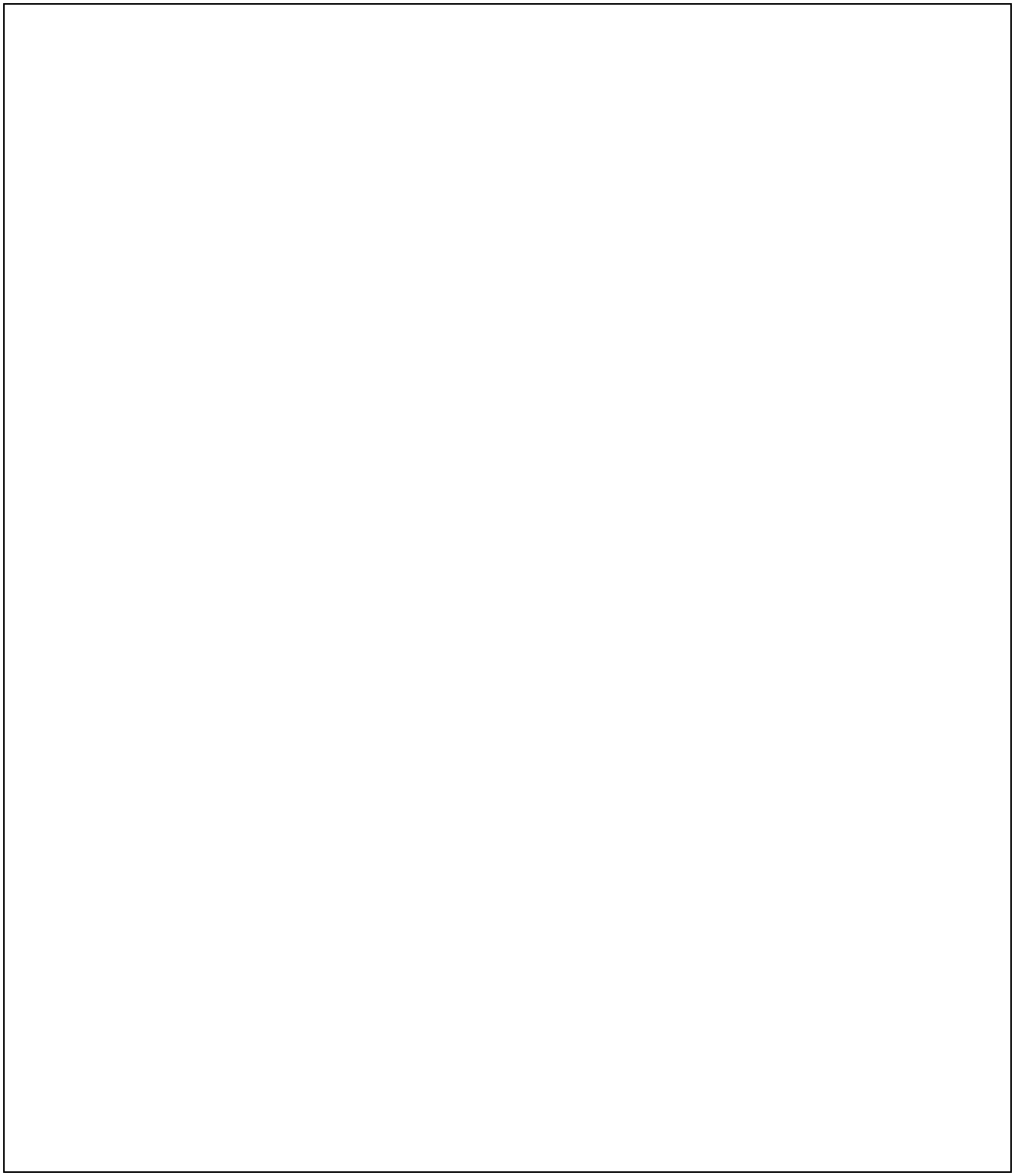
Proposez une modification de vecteur vector2D permettant de rendre accessible ces deux champs.

Question n° 8.3:

Est-ce que les opérations « + », « - », « += », « -= », « * » héritées de vector2D dans vector3D sont pertinentes dans la définition de vector3D ?

Expliquer pourquoi ?

Qu’est-il nécessaire de faire pour avoir les opérations attendues pour vector3D ?



Question n° 8.4:

La fonction « norm » définie précédemment doit être redéfinies dans la nouvelle classe vector3D. Nous souhaitons que cette nouvelle définition remplace définitivement la fonction « norm » définie dans la classe c. Comment faut-il faire pour être certain que le code suivant retourne bien la norme du vecteur vector3D $\sqrt{x^2 + y^2 + z^2}$ et non la norme $\sqrt{x^2 + y^2}$ telle que calculée par la fonction « norm » implantée par la classe vector2D :

```
template<typename floatT>
void print_norm(std::vector2D<floatT>& aVector)
{
    std::cout << "Norm:" << aVector.norm() << std::end;
}

int main()
{
    std::vector3D v3D(1, 2, 1);  // Doit affichée
    print_norm(v3D);
}
```