

# PROSIT ALLER

## Contexte

Un type a fait un programme pour créer des points en 2D puis il essaie avec des points en 3D mais il rencontre des difficultés avec la structure du code :

Redondances et similitudes

Iel n’arrive pas à passer un point 3D en 2D

## MOTS CLES

Parcours

2D et 3D

Héritage

ABST

Classe

Virtu

Poly

Redondance

Abstract

## PROBLEMATIQUE

**Comment passer d’un point 3D en 2D et inversement ?**

**Comment restructurer le code pour éviter les redondances ?**

## Contraintes :

Coordonnées à enlever Z.

Pas de redondance

Structure du main (photos programme)

## Livrables :

Diagramme de classe

Programme

## Généralisation :

Savoir implémenter l’héritage (en tuant ses grands-parents)

## Piste de solutions :

Constructeur vide

Créer classe avec héritage

Modifier partie « parcours » du code

Enlever coordonnées Z

## Plan d’action :

Faire le diagramme de classe

Conception du code -> classe en se basant sur le main

Conclure

## Réalisation du plan d’action

Parcours : Chemin pour aller d’un point à un autre

2D et 3D : 2 dimensions (x,y), 3 dimensions (x,y,z)

Heritage : l’héritage est un mécanisme qui permet, lors de la déclaration d’une nouvelle classe, d'y inclure les caractéristiques d’une autre classe.

Classe : En programmation orientée objet, une classe déclare des propriétés communes à un ensemble d'objets.

Virtu : Une fonction virtuelle est un membre dont vous souhaitez la redéfinition dans des classes dérivées. En faisant référence à un objet de classe dérivée à l’aide d’un pointeur ou d’une référence à la classe de base, on peut appeler une fonction virtuelle depuis cet objet et exécuter la version de la classe dérivée.

Poly : plusieurs en latin

Redondance : répéter une information sous plusieurs formes sans que cela soit nécessaire

Abstract : expressions de concepts généraux dont des classes plus spécifiques peuvent être dérivées. On ne peut pas créer d’objet d’un type de classe abstraite.

On s’est ensuite penché sur la realisation du code avec matthieu

Main :

1. #include <iostream>
2. #include "Point3D.h"
3. #include "Point2D.h"
4. #include "Parcours2D.h"
5. #include "Parcours3D.h"
7. int main()
8. {
9. Point2D\* A = new Point2D(11, 4.8);
10. Point2D\* B = new Point2D(58, 6);
11. Point2D\* C = new Point2D(78, 52);
13. Parcours2D\* parcours1 = new Parcours2D();
15. parcours1->ajouter(\*A);
16. parcours1->ajouter(\*B);
17. parcours1->ajouter(\*C);
19. parcours1->distance();
21. Point3D\* D = new Point3D(9, 78, 5);
22. Point3D\* E = new Point3D(6, 45, 12);
23. Point3D\* F = new Point3D(95, 24, 36);
25. Parcours3D\* parcours2 = new Parcours3D();
27. parcours2->ajouter(\*D);
28. parcours2->ajouter(\*E);
29. parcours2->ajouter(\*F);
31. parcours2->distance();
33. Point2D\* G = new Point2D(0,0);
34. Point3D\* H = new Point3D(1,1,1);
36. (H = G)->afficherCoordo();
37. //(G = H)->afficherCoordo();
38. }

Point3D.h

1. class Point3D {
2. protected:
3. float x;
4. float y;
5. float z;
6. public:
7. Point3D();
8. Point3D(float m\_x, float m\_y, float m\_z);
9. virtual void distance(const Point3D& autre);
10. virtual void afficherCoordo();
11. virtual float getX();
12. virtual float getY();
13. virtual float getZ();
14. void retirerZ();
15. };

Point3D.cpp

1. #include "Point3D.h"
2. #include <iostream>
3. #include <math.h>
5. using namespace std;
7. Point3D::Point3D(){};
9. Point3D::Point3D(float m\_x, float m\_y, float m\_z) : x(m\_x), y(m\_y), z(m\_z) {};

12. void Point3D::distance(const Point3D& autre) {
13. double dx, dy, dz;
14. dx = x - autre.x;
15. dy = y - autre.y;
16. dz = z - autre.z;
18. float distance = sqrt(dx \* dx + dy \* dy + dz \* dz);
20. cout << "Distance : " << distance << "\n";
21. };
23. void Point3D::afficherCoordo() {
24. cout << "X : " << x << "\n";
25. cout << "Y : " << y << "\n";
26. cout << "Z : " << z << "\n";
27. };
29. void Point3D::retirerZ() {
30. z = 0;
31. }
33. float Point3D::getX() {
34. return x;
35. };
36. float Point3D::getY() {
37. return y;
38. };
39. float Point3D::getZ() {
40. return z;
41. };

Point2D.h

1. #include "Point3D.h"
3. class Point2D : public Point3D {
4. public:
5. Point2D();
6. Point2D(float m\_x, float m\_y);
7. void ajouterZ(float m\_z);
8. //void operator=(const Point3D& b);
9. };

Point2D.cpp

1. #include "Point2D.h"
3. Point2D::Point2D() {
4. z = 0;
5. };
7. Point2D::Point2D(float m\_x, float m\_y) {
8. x = m\_x;
9. y = m\_y;
10. z = 0;
11. };
13. void Point2D::ajouterZ(float m\_z) {
14. z = m\_z;
15. }
17. //void Point2D::operator=(const Point3D& b) { x = b.x; y = b.y; z = 0; };

Parcours3D.h

1. #include <iostream>
2. #include <vector>
3. #include "Point3D.h"
5. class Parcours3D {
6. private:
7. std::vector<Point3D> listepoints;
8. public:
9. Parcours3D();
10. Parcours3D(Point3D& point);
11. ~Parcours3D();
12. void ajouter(Point3D& point);
13. void distance();
14. };

Parcours3D.cpp

1. #include "Parcours3D.h"
3. using namespace std;
5. Parcours3D::Parcours3D() {
6. };
8. Parcours3D::Parcours3D(Point3D& point) {
9. listepoints.push\_back(point);
10. };
12. void Parcours3D::ajouter(Point3D& point) {
13. listepoints.push\_back(point);
14. };
16. void Parcours3D::distance() {
17. float distance = 0;
18. for (int i = 0; i < listepoints.size() - 1; i++)
19. {
20. double dx, dy, dz;
21. dx = listepoints[i].getX() - listepoints[i + 1].getX();
22. dy = listepoints[i].getY() - listepoints[i + 1].getY();
23. dz = listepoints[i].getZ() - listepoints[i + 1].getZ();
25. distance += sqrt(dx \* dx + dy \* dy + dz \* dz);
26. }
27. cout << "Distance parcours : " << distance << "\n";
28. }
30. Parcours3D::~Parcours3D() {};

Parcours2D.h

1. #include <iostream>
2. #include <vector>
3. #include "Point2D.h"
4. #include "Point3D.h"
6. class Parcours2D {
7. private:
8. std::vector<Point2D> listepoints;
9. public:
10. Parcours2D();
11. Parcours2D(Point2D& point);
12. ~Parcours2D();
13. void ajouter(Point2D& point);
14. void distance();
15. };

Parcours2D.cpp

1. #include "Parcours2D.h"
3. using namespace std;
5. Parcours2D::Parcours2D() {
6. };
8. Parcours2D::Parcours2D(Point2D& point) {
9. listepoints.push\_back(point);
10. };
12. void Parcours2D::ajouter(Point2D& point) {
13. listepoints.push\_back(point);
14. };
16. void Parcours2D::distance() {
17. float distance = 0;
19. for (int i = 0; i < listepoints.size() - 1; i++)
20. {
21. float dx, dy;
22. dx = listepoints[i].getX() - listepoints[i + 1].getX();
23. dy = listepoints[i].getY() - listepoints[i + 1].getY();
25. distance += sqrt(dx \* dx + dy \* dy);
26. }
27. cout << "Distance parcours : " << distance << "\n";
28. }
30. Parcours2D::~Parcours2D() {};