Programmation orientée objet

LIV 2024-2025

Travaux pratiques 4

Site du cours : `https://defelice.up8.site/poo.html`
Les exercices marqués de (@) sont optionnels.
Un fichier écrit en langage C++ se termine conventionnellement par .cpp.
Une commande de compilation est `g++ fichier_source1.cpp fichier_source2.cpp fichier_source3.cpp`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall` pour demander au compilateur d'afficher plus d'avertissements (warnings)
- `-Wextra` pour demander au compilateur d'afficher plus d'avertissements (warnings)
- `-std=c++20` pour compiler selon la norme c++20
- `-g -fsanitize=address` pour que l'exécutable puisse detecter certaines erreurs de mémoire (executable plus lent, -g pour avoir plus de précision).
- `-g` pour ajouter des informations de débugage, utile avec debug (executable plus lent).

Exemple : `g++ -Wall -Wextra fichier1.cpp -o monprogramme`

Exercice 1. *Tableau 2*

Implanter une classe `Tableau2` qui améliore la classe `Tableau` (ou la classe `TableauExt`) avec l'opérateur `[]`. Les classes `Tableau` et `TableauExt` viennent d'un exercice de TP précédent.

```
Tableau2 tab1{20};

tab1[10]=5;
int a=tab1[10];
// int b=tab1[30]; à l'exécution message d'erreur, tab1 n'a pas de case 30
```

Exercice 2. *Séparation*

En vous inspirant de la classe `Point2D` (contenue dans `annexe03.cpp`, voir ci-dessous).

annexe03.cpp

```
/* Attention cette classe bien que synaxiquement
 * correcte aurait pu être mieux conçue.
 * Elle représente des points du plan */

#include<cstdio>
#include<cstdlib>
class Point2D
{
   double* xy=nullptr;
      // les deux coordonnées sont
      // stockée dans un tableau
      // de 2 <double>
   public :
   Point2D(double x,double y)
   {
      xy=(double*)malloc(sizeof(double)*2);
      if(xy==nullptr) { exit(1); }
      xy[0]=x;
      xy[1]=y;
   }
   double getX(){return *xy;}
```

```
22     double getY(){return *(xy+1);}
23     Point2D(const Point2D& p){
24        xy=(double*)malloc(sizeof(double)*2);
25        if(xy==nullptr) { exit(1); }
26        xy[0]=p.xy[0];
27        xy[1]=p.xy[1];
28     }
29     ~Point2D() { free(xy); }
30     const Point2D& operator = (const Point2D& p)
31     {
32        xy[0]=p.xy[0];
33        xy[1]=p.xy[1];
34        return *this;
35     }
36     void rotate(void){ // fait tourner le point d'un quart
37                        // de tour autour du point (0,0)
38        double t=xy[0];
39        xy[0]=-xy[1];
40        xy[1]=t;
41     }
42     void affiche(void) {
43        printf("(%5f,%5f)\n",xy[0],xy[1]);
44     }
45  };
46  int main()
47  {
48        Point2D a{4,2};
49        a.rotate();
50        a.affiche();  // doit afficher le point -2,4
51  }
52
```

Construisez la classe `Point2D` qui représente des points du plan. La classe `Point2D` doit contenir les méthodes `affiche` et `rotate` et le constructeur `Point2D(double,double)`. Elle doit être séparée en deux fichiers `point2d.cpp` et `point2d.hpp`.

Le fichier `point2d.hpp` contient la définition de la classe. Le fichier `point2d.hpp` ne doit pas contenir d'instruction, toutes les méthodes doivent être définies dans `point2d.cpp`. Exemple du contenu de `main.cpp` et commande de compilation.

```
#include"point2d.hpp"
int main(){
   Point2d a{4,2};
   a.rotate();
   a.affiche();  // doit afficher le point -2,4
}
```

compilation :

```
g++ -c [autres options] main.cpp
g++ -c [autres options] point2d.cpp
g++ [options] main.o point2d.o
```

Exercice 3. *Figure*

Fabriquer la classe `Figure` qui modèlise des figures du plan. Ici une figure du plan est simplement une liste ordonnée de `Point2D` (de l'exercice séparation).La classe `Figure` possède deux méthodes `affiche` et `rotate` qui utilisent les méthodes `affiche` et `rotate` de la classe `Point2D`. Il ne doit pas y avoir de limite maximum de

points à la taille d'une figure. Il faut écrire la classe `figure` dans deux fichiers `figure.hpp` et `figure.cpp`. Pensez à bien gérer la mémoire, pour vérifiez utiliser l'option `-fsanitize=address`.

```
#include"figure.hpp"
#include"point2d.hpp"
int main(){
   Figure o; // figure vide
   o.ajouter(Point2D{0.,1.});
   o.ajouter(Point2D{1.,2.});
   o.rotate(); // rotation de 90 d à chaque point de la figure o
   o.affiche(); // affiche (textuellement) la liste des points
}
```

Exercice 4. *Figure et Point2D amélioration*

Cet exercice consiste simplement à améliorer (en parallèle) les classes `Figure` et `Point2D` afin de permettre aux figures d'effectuer d'autres opérations. Améliorations demandées :

- ajouter à `Figure` la méthode d'objet `translater(Point2D a)` qui translate chaque point selon le vecteur $\overrightarrow{O\texttt{a}}$.
  Où O est le point de coordonnées $(0, 0)$
- ajouter à `Figure` la méthode d'objet `dilater(double d)` qui multiplie les coordonnées de chaque point par `d`.
- ajouter à la classe `Figure` la méthode d'objet `tourner(double a)` qui effectue une rotation de `a` degrés à chaque point de la figure autour du point de coordonnées $(0, 0)$.