Programmation orientée objet

LIV 2025-2026

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 (adresse inacessible, fuite mémoire) (l'inonveniant et que l'executable est 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).

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"
#include"point2d.hpp"  // oui, deux fois, pour vérifier protection inclusion multiple
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
```

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
      // mémorisées 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;}
   double getY(){return *(xy+1);}
   Point2D(const Point2D& p){
      xy=(double*)malloc(sizeof(double)*2);
      if(xy==nullptr) { exit(1); }
      xy[0]=p.xy[0];
      xy[1]=p.xy[1];
   }
   ~Point2D() { free(xy); }
   const Point2D& operator = (const Point2D& p)
   {
      xy[0]=p.xy[0];
      xy[1]=p.xy[1];
      return *this;
   }
   void rotate(void){ // fait tourner le point d'un quart
                      // de tour autour du point (0,0)
      double t=xy[0];
      xy[0]=-xy[1];
      xy[1]=t;
   }
   void affiche(void) {
      printf("(%5f,%5f)\n",xy[0],xy[1]);
   }
};
int main()
{
      Point2D a{4,2};
      a.rotate();
      a.affiche();  // doit afficher le point -2,4
}

```

Exercice 3. *Constant*

Créer la classe `MemInt` dont chaque objet mémorise une valeur entière.

```
MemInt n{4};
// MemInt m; erreur  de compilation il faut un argument !
n.get(); // renvoie 4
n.set(5);
n.get(); // renvoie 5
const MemInt k{2};
k.get(); // renvoie 2
// k.set(2); // erreur de compilation ! interdit car k est const
```

Exercice 4. *Nombre acces*

Créer la classe `ObjAcces` qui mémorise un nombre entier, et compte les accès à cette valeur depuis la dernière modification. Puis (dans un second temps) compléter la classe pour qu'elle puisse gérer les objets constants.

```
ObjAcces n{4};
n.nbAcces(); // renvoie 0
n.get(); // renvoie 4
n.nbAcces(); // renvoie 1
n.nbAcces(); // renvoie 1
n.get(); // renvoie 4
n.nbAcces(); // renvoie 2
n.set(2);
n.nbAcces(); // renvoie 0
n.get(); // renvoie 2
n.nbAcces(); // renvoie 1
ObjAcces r{2};
r.nbAcces(); // renvoie 0
n.nbAcces(); // renvoie 1

ObjAcces const c{5};
c.nbAcces(); // renvoie 0
c.get(); // renvoie 5
c.nbAcces(); // renvoie 1
c.nbAcces(); // renvoie 1
c.get(); // renvoie 5
c.get(); // renvoie 5
c.nbAcces(); // renvoie 3
//c.set(4);  erreur de compilation c est constant
```