Programmation orientée objet

LIV 2025-2026

Travaux Dirigé 1

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. *Intervalle 1*

Le but de cet exercice est d'implanter une classe pour représenter les intervalles fermés bornés.

Intervalle (fermé borné)

Un intervalle fermé borné est un sous-ensemble de nombres réels que l'on note (souvent) $[a; b]$ où a et b sont des nombres réels avec $a \leq b$. Les valeurs a et b sont appelées respectivement l' extrémité inférieure et l'extrémité supérieure de l'intervalle. Par exemple, $[-2; 4]$ et [-1,3 ; 4,98] sont deux intervalles (fermés bornés).

Un nombre n appartient à l'intervalle...

- s'il est égal à l'une des deux extrémités
- s'il est plus grand que l'extremité inférieure et plus petit que l'extrémité supérieure (autrement dit compris entre les extrémités).

L'ensemble vide est un intervalle et $[3; 3]$ est un intervalle qui ne contient qu'un seul élément : 3.

Exemple :

. 3,5 appartient à $[2; 4]$,
. 2 appartient à $[2; 4]$,
. 4 appartient à $[2; 4]$,
. mais 4,5 n'appartient pas à $[2; 4]$.

Les intervalles étant des ensembles on peut considérer l'intersection ($\cap$) ou l'union ($\cup$) d'intervalles. Une intersection d'intervalle est toujours un intervalle ce n'est pas le cas pour l'union. Par exemple $[1; 3] \cap [2; 4] = [2; 3]$. Mais $[-2; -1] \cup [1; 2]$ n'est pas un intervalle.

But de l'exercice

Implanter en `C++` une classe `Interv` dont chaque objet (i.e instance) modélise un intervalle fermé borné, pour les extrémités on utilisera des valeurs de types `double`.

La classe `Interv` doit contenir les méthodes suivantes :

- Un constructeur avec deux paramètres `double` (les deux extrémités de l'intervalle),
- un constructeur sans paramètre (qui construit l'ensemble vide),
- un constructeur avec un paramètre `a` (qui contruit $[a; a]$ un intervalle contenant un seul élément),
- une méthode `appartient(double e)` qui renvoie 1 (ou `true`) si le nombre `e` est dans l'intervalle, 0 sinon,

Exemple :

```
int main(){
   Interv i1{1,2}; // l'intervalle [1;2]
   i1.appartient(1.2); // renvoie 1
   i1.appartient(0.3); // renvoie 0
```

```
i1.appartient(2); // renvoie 1
i1.appartient(2.01); // renvoie 0

Interv i2{4}; // [4;4] ensemble ne contenant que 4
i2.appartient(4);// renvoie 1
i2.appartient(4.001);// renvoie 0
i2.appartient(3.999);// renvoie 0

Interv i3; // i3 est vide
double a= (double)rand()/100000; // valeur aléatoire
i3.appartient(a); // renvoie 0 (pour n'importe quelle valeur de a
```

Exercice 2. *Intervalle 2*

A la classe `Interv` ajoutez les méthode suivantes.

— une méthode `estVide(void)` qui renvoie 1 si l'intervalle est l'ensemble vide, et 0 sinon.

— une méthode `Interv intersec(Interv b)` qui construit et renvoie l'intersection de deux intervalles,

— une méthode `unifier` qui renvoie l'union de deux intervalles seulement si cette union est elle-même un intervalle. Dans le cas contraire on peut interrompre le programme par exemple avec `exit(1);`.