ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communications
Cours de Programmation Orientée Objet, section PH
Chappelier J.-C.

NOM : Hanon Ymous
(000000)
Place : 0

#0000

Programmation Orientée Objet (SPH) : Examen final

22 mai 2025

INSTRUCTIONS (à lire attentivement)

IMPORTANT ! Veuillez suivre les instructions suivantes à la lettre sous peine de voir votre examen annulé dans le cas contraire.

1. Vous disposez d'une heure quarante-cinq minutes pour faire cet examen (9h15 – 11h00).
2. Vous devez **écrire à l'encre noire ou bleu foncée**, pas de crayon ni d'autre couleur.

   N'utilisez **pas non plus de stylo effaçable** (perte de l'information à la chaleur).
3. Vous avez droit à toute documentation papier.

   En revanche, vous ne pouvez pas utiliser d'ordinateur personnel, ni de téléphone portable, ni aucun autre matériel électronique.
4. Répondez aux questions directement sur la donnée ; ne joignez aucune feuille supplémentaire ; **seul ce document sera corrigé**.
5. Lisez attentivement et *complètement* les questions de façon à ne faire que ce qui vous est demandé. Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un(e) des assistant(e)s.
6. L'examen comporte quatre exercices indépendants, qui peuvent être traités dans n'importe quel ordre, mais qui ne rapportent pas la même chose (les points sont indiqués, le total est de 110) :
   1. deux petites questions : 10 points ;
   2. conception et programmation 1 : 40 points ;
   3. correction de code : 20 points ;
   4. conception et programmation 2 : 40 points.

   Tous les exercices comptent pour la note finale.

Question 1 – Faisons le point [10 points]

1.1 Trois petits codes [5 points]

Pour chacun des trois codes proposés ci-dessous, dites :

1. s'il compile ;
2. si oui, s'il s'exécute correctement ; (et sinon laissez blanc ;)
3. si oui aux deux questions précédentes : ce qu'il affiche ;
   et si non à l'une des deux questions précédentes : la cause de l'erreur ;
4. s'il y a une fuite de mémoire ; et si oui : sur qui ?/pourquoi ?

Note : on **ne** s'intéresse **pas** au style ni aux instructions possiblement inutiles.

```cpp
#include <iostream>
using namespace std;

int main()
{
  double a(12.34);
  double* ptr(&a);
  cout << *ptr << endl;
  ptr = nullptr;
  return 0;
}
```

code1.cc

```cpp
#include <iostream>
using namespace std;

class A {
public:
  A(double x)
  : ptr(new double(x))
  {}
  ~A() { ptr = nullptr; }
  void print() const
  { cout << *ptr << endl; }
private:
  double* ptr;
};

int main()
{
  A a(5.67);
  a.print();
  return 0;
}
```

code2.cc

```cpp
#include <iostream>
#include <memory>
using namespace std;

class A {
public:
  A(double x)
  : ptr(make_unique<double>(x))
  {}
  void print() const
  { cout << *ptr << endl; }
private:
  unique_ptr<double> ptr;
};

int main()
{
  A a(89.1);
  a.print();
  return 0;
}
```

code3.cc

	code1.cc	code2.cc	code3.cc
compile ? (oui/non)			
s'exécute ? (oui/non)			
affichage ou cause de l'erreur			
fuite mémoire ? (oui/non) + brève explication			

1.2 C'est fini [5 points]

Qu'affiche le code suivant ? **Justifiez** votre réponse.

```
 1 #include <iostream>
 2 using namespace std;
 3
 4 class A {
 5 public:
 6   virtual ~A() = default;
 7   virtual double f(int a) const = 0;
 8 };
 9
10 class B : public A {
11 public:
12   double f(int b) const {
13     if (b > 10) return 1.23;
14     return 98.76;
15   }
16 };
17
18 class C : public B {
19 public:
20   ~C() { cout << "Capri" << endl; }
21
22   double f(int c) const {
23     if (c > 100) return 5.65;
24     return -33.3;
25   }
26 };
27
28 int main()
29 {
30   C ta_c;
31   B* luga(&ta_c);
32   cout << luga->f( 25 ) << endl;
33   return 0;
34 }
```

Réponse et justification :

suite au dos ☞

Question 2 – Propriétaires et locataires [40 points]

2.1 Cadre général [20 points]

On souhaite écrire un programme permettant de représenter des habitations ayant des propriétaires et des locataires. On dispose pour cela d'une classe `Personne` et d'une classe `Habitation`, dont une portion de code est donnée au dos. Un exemple illustratif de `main()` est aussi donné à la suite.
Un *exemple* de résultat possible est également fourni ci-dessous.

On souhaite que les `Personne` aient en plus :
- leurs logements : la liste des habitations où elles logent (une même personne peut avoir plusieurs logements) ;
- leurs propriétés : la liste des habitations dont elles sont propriétaires ;
- une méthode `habite()` pour ajouter une habitation à leurs logements ;
- une méthode `possede()` pour ajouter une habitation à leurs propriétés.

La classe `Personne` pourra avoir aussi d'autres méthodes suivant vos besoins pour la suite, mais on ne vous demande strictement *rien* concernant les affichages. Tout ce qui concerne l'affichage serait fait par une autre programmeuse.

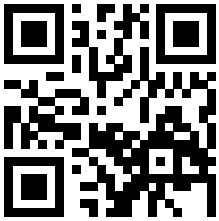
De leur coté, les habitations doivent nécessairement avoir un propriétaire : une personne connue à l'avance et qui ne change pas. Elles doivent aussi avoir la liste de leurs habitants, ainsi qu'une méthode `loge()` qui permet d'ajouter un habitant.

Simplification : pour *tous* les ajouts dans des listes (logements, propriétés ou habitants), on **ne** vous demande **pas** de vérifier si l'élément à ajouter y est déjà présent.

Après le code fourni, **complétez la définition de ces deux classes** en précisant à chaque fois dans laquelle des cinq zones indiquées (A à E) vous écririez ce code.

Exemple de sortie possible pour le `main()` fourni :

```
Propriétaires :
- Julie qui possède 3 propriétés : <maison1>, <immeuble>, <chalet>
  et habite dans 1 habitation : <maison1>

- Sylvie qui possède 1 propriété : <maison2>
  et habite dans 1 habitation : <immeuble>

Locataires :
- Pierre qui ne possède rien
  et habite dans 1 habitation : <immeuble>

- Jean qui ne possède rien
  et habite dans 2 habitations : <chalet>, <maison2>

- Sylvie qui possède 1 propriété : <maison2>
  et habite dans 1 habitation : <immeuble>

Habitations :
<maison1>  appartenant à Julie  et ayant 1 habitant  : Julie
<immeuble> appartenant à Julie  et ayant 2 habitants : Pierre, Sylvie
<chalet>   appartenant à Julie  et ayant 1 habitant  : Jean
<maison2>  appartenant à Sylvie et ayant 1 habitant  : Jean
```



```
#include <iostream>
#include <vector>
#include <string>
#include <string_view>
using namespace std;

// à vous de compléter ici si nécessaire (zone A)

// ---------------------------------------------------------------------
class Personne {
public:
  Personne(string_view nom) : nom_(nom) {}

  string nom() const { return nom_; }

  void affiche() const { // serait fait par qq1 d'autre  }

  // à vous de compléter ici (zone B)

private:
  const string nom_;

  // à vous de compléter ici si nécessaire (zone C)
};

// ---------------------------------------------------------------------
class Habitation
{
public:
  void affiche() const { // serait fait par qq1 d'autre  }

  // à vous de compléter ici (zone D)

private:
  // à vous de compléter ici si nécessaire (zone E)
};

// ---------------------------------------------------------------------

// à vous de compléter ici si nécessaire (zone F)

// ---------------------------------------------------------------------
vector<const Personne*> proprietaires(vector<Personne> const& liste)
{
  // À faire plus loin (voir la suite de la donnée)
}

vector<const Personne*> locataires(vector<Personne> const& liste)
{
  // À faire plus loin (voir la suite de la donnée)
}

// ... suite au dos ...
```

```
// ...suite...
void affiche(vector<const Personne*> const& liste)
{ // fait par quelqu'un d'autre  }

// ----------------------------------------------------------------------
int main()
{
  vector<Personne> population({
    Personne("Julie"), Personne("Pierre"), Personne("Jean"), Personne("Sylvie")
  });
  Personne& julie = population[0];        Personne& pierre = population[1];
  Personne& jean  = population[2];        Personne& sylvie = population[3];

  Habitation maison1 (julie);             Habitation maison2(sylvie);
  Habitation immeuble(julie);
  Habitation chalet  (julie);

  // habite chez soi
  julie.habite(maison1);

  // locataire
  pierre.habite(immeuble);

  // propriétaire locataire
  sylvie.habite(immeuble);

  // plusieurs résidences
  jean.habite(chalet);   jean.habite(maison2);

  cout << "Propriétaires :" << endl;
  affiche(proprietaires(population))

  cout << "Locataires :" << endl;
  affiche(locataires(population));

  cout << "Habitations :" << endl;
  maison1.affiche(); immeuble.affiche(); chalet.affiche(); maison2.affiche();

  return 0;
}
```

Début de réponse (n'oubliez pas de préciser dans quelle zone (A–E) vous ajoutez votre code) **:**

Suite de la réponse (n'oubliez pas de préciser dans quelle zone (A–E) vous ajoutez votre code) **:**

suite au dos ☞

2.2 Lister les propriétaires et les locataires d'une population [15 points]

On souhaite maintenant fournir deux fonctions (`proprietaire()` et `locataires()` ; voir le code fourni précédemment) qui permettent, à partir d'une liste de personnes, d'extraire, l'une, la sous-liste des propriétaires présent dans la liste reçue en paramètre, et, l'autre, la sous-liste des locataires.

Ces sous-listes sont bien sûr des listes de *références* vers des personnes existant dans la population de départ, d'où le type de retour proposé dans le code fourni précédemment.

Donnez ici la définition de chacune de ces deux fonctions. Ajoutez si nécessaire des éléments aux classes définies précédemment (précisez quel code et dans quelle zone).

Réponses :

2.3 Achats et ventes [5 points]

Si l'on souhaitait pouvoir vendre des habitations, c.-à-d. les faire changer de propriétaire, **est-ce que cela changerait l'implémentation** que vous avez faite jusqu'ici ?

Si *oui*, **dites en quoi** (c.-à-d. où) et **pourquoi**.
Si c'est *non*, écrivez la méthode `vente()` des habitations, laquelle ferait changer leur propriétaire.

Réponse :

suite au dos ☞

Question 3 – Algèbre linéaire [20 points]

Un élève envisage de faire un projet de programmation d'outils d'algèbre linéaire en C++. Il commence pour cela par mettre en place un cadre minimal factice, presque vide, pour tester l'organisation prévue pour ses fichiers. Il a donc produit les fichiers suivants (tels quels, avec les commentaires) :

`Makefile` :

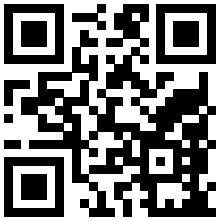
```
1  CC = $(CXX)
2
3  all: algebre_lineaire
4
5  algebre_lineaire: algebre_lineaire.o matrice.o vecteur.o
6
7  affichable.o: affichable.cc affichable.h
8  algebre_lineaire.o: algebre_lineaire.cc matrice.h affichable.h vecteur.h
9  matrice.o: matrice.cc matrice.h affichable.h vecteur.h
10 vecteur.o: vecteur.cc affichable.h vecteur.h matrice.h
```

`affichable.h` :

```
1  #pragma once
2  #include <iostream>
3
4  class Affichable {
5  public:
6    virtual void affiche(ostream&) const = 0;
7  };
8
9  ostream& operator<<(ostream& out, Affichable const& a);
```

`matrice.h` :

```
1  #pragma once
2
3  #include "affichable.h"
4  #include "vecteur.h"
5
6  class Matrice : public Affichable {
7  public:
8    // ...other things...
9
10   Vecteur operator*(Vecteur const& v);
11
12   virtual void affiche(std::ostream&) const override;
13
14 private:
15   // whatever...
16 };
17
18 class MatriceUnitaire : public Matrice {
19   // ...something...
20 };
```



vecteur.h :

```
 1 #pragma once
 2
 3 #include "affichable.h"
 4 #include "matrice.h"
 5
 6 class Vecteur : public Affichable {
 7 public:
 8   // ...other things...
 9
10   Vecteur rotation(MatriceUnitaire const& m) const;
11
12   virtual void affiche(std::ostream&) const override;
13
14 private:
15   // whatever...
16 };
17
```

affichable.cc :

```
 1 #include "affichable.h"
 2
 3 #include <iostream>
 4 using namespace std;
 5
 6 ostream& operator<<(ostream& out, Affichable const& a)
 7 {
 8   a.affiche(out);
 9   return out;
10 }
```

matrice.cc :

```
 1 #include "matrice.h"
 2 #include "vecteur.h"
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 Vecteur Matrice::operator*(Vecteur const& v)
 8 {
 9   cout << "mat mult vect" << endl;
10   return Vecteur();
11 }
12
13 void Matrice::affiche(ostream& out) const
14 { out << "matrice"; }
```

vecteur.cc :

```
 1 #include "affichable.h"
 2 #include "vecteur.h"
 3 #include "matrice.h"
 4
 5 #include <iostream>
 6 using namespace std;
 7
 8 Vecteur Vecteur::rotation(
 9                 MatriceUnitaire const& m) const
10 {
11   cout << "rotation de " << *this
12        << " par " << m << endl;
13   return m * (*this);
14 }
15
16 void Vecteur::affiche(ostream& out) const
17 { out << "vecteur"; }
```

Ainsi que le fichier `algebre_lineaire.cc`, qui contient le `main()`. Il n'est pas donné ici car il n'est pas lui-même source d'erreur directe.

Une première tentative de compilation lui donne plein d'erreurs, mais dont voici les deux essentielles :

```
g++    -c -o algebre_lineaire.o algebre_lineaire.cc

In file included from matrice.h:3,
                 from algebre_lineaire.cc:1:
affichable.h:6:24: error: 'ostream' has not been declared
    6 |   virtual void affiche(ostream&) const = 0;
      |                        ^~~~~~~

In file included from matrice.h:4:
vecteur.h:10:20: error: 'MatriceUnitaire' has not been declared
   10 |   Vecteur rotation(MatriceUnitaire const& m) const;
      |                    ^~~~~~~~~~~~~~~
```

À noter que la seconde erreur aurait aussi très bien pu être :

```
In file included from vecteur.h:4,
                 from algebre_lineaire.cc:2:
matrice.h:9:3: error: 'Vecteur' does not name a type
    9 |   Vecteur operator*(Vecteur const& v) const;
      |   ^~~~~~~
```

Que doit-il faire pour corriger ces **deux** erreurs ?

Apportez **directement vos corrections** sur le code donné précédemment et **indiquez ici** les lignes de quels fichiers vous avez corrigées :

Lignes corrigées :

Après avoir suivi vos conseils et corrigé les deux erreurs ci-dessus, il en a une troisième :

```
g++    -c -o vecteur.o vecteur.cc
vecteur.cc: In member function 'Vecteur Vecteur::rotation(const MatriceUnitaire&) const':
vecteur.cc:11:20: error: passing 'const MatriceUnitaire' as 'this' argument
                         discards qualifiers [-fpermissive]
   11 |   return m * (*this);
      |                    ^
In file included from vecteur.cc:3:
matrice.h:10:11: note:   in call to 'Vecteur Matrice::operator*(const Vecteur&)'
   10 |   Vecteur operator*(Vecteur const& v);
      |           ^~~~~~~~
make[1]: *** [<builtin>: vecteur.o] Error 1
```

Que doit-il faire pour corriger cette erreur ?

Apportez **directement vos corrections** sur le code donné précédemment et **indiquez ici** les lignes de quels fichiers vous avez corrigées :

Lignes corrigées :

Il parvient donc finalement à compiler et aboutit à l'erreur suivante :

```
g++    -c -o algebre_lineaire.o algebre_lineaire.cc
g++    -c -o matrice.o matrice.cc
g++    -c -o vecteur.o vecteur.cc
g++   algebre_lineaire.o matrice.o vecteur.o   -o algebre_lineaire
/usr/bin/ld: vecteur.o: in function `Vecteur::rotation(MatriceUnitaire const&) const':
vecteur.cc:(.text+0x3b): undefined reference to `operator<<(std::ostream&, Affichable const&)'
/usr/bin/ld: vecteur.cc:(.text+0x62): undefined reference to
                                      `operator<<(std::ostream&, Affichable const&)'
collect2: error: ld returned 1 exit status
make[1]: *** [<builtin>: algebre_lineaire] Error 1
```

Que doit-il faire pour corriger cette erreur ?

Apportez **directement vos corrections** sur les fichiers donnés précédemment et **indiquez ici** les lignes de quels fichiers vous avez corrigées :

Lignes corrigées :

suite au dos ☞

Question 4 – Personnages et lieux [40 points]

On souhaite écrire une toute petite partie d'un programme de jeu ayant divers types de personnages (marchands, moines, ...) et divers types de lieux (magasins, bibliothèques, ...).
Les personnages ont la capacité de faire une action pour un lieu donné, et les lieux peuvent déclencher cette capacité pour un personnage donné. Par exemple, un marchand pour un magasin va faire une action de vente, un moine pour une bibliothèque l'action d'écrire, etc.

4.1 Marchands, moines, magasins et bibliothèques [25 points]

Définissez ci-dessous et sur la page suivante le code C++ permettant de mettre en œuvre la description précédente.
Concrètement, donnez tout le code nécessaire pour avoir des marchands et des moines (personnages), ainsi que des magasins et des bibliothèques (lieux). Pour simplifier, l'action d'un personnage dans un lieu sera simplement d'afficher un mot :

- un marchand pour un magasin affichera simplement `"vend"` ;
- un marchand pour une bibliothèque : `"lit"` ;
- un moine pour un magasin : `"achète"` ;
- un moine pour une bibliothèque : `"écrit"` ;

Remarque importante : même si c'est ici simplifié à l'extrême, il faut bien considérer ces actions comme de *vraies actions* qui devraient en réalité être bien plus compliquées et utiliseraient pleinement les caractéristiques de chaque personnage et de chaque lieu.

Réponse :

Ne pas écrire dans cette zone.

suite au dos ☞

4.2 Le jeu [15 points]

Définissez ci-dessous la classe `Jeu` (uniquement pour les aspects présentés dans le paragraphe d'introduction). On ajoutera simplement une méthode `tour()` qui permet de lancer (une fois) l'action de chaque personnage pour chaque lieu du jeu (cf exemple à la fin de la section suivante).

Réponse :

Complétez enfin le `main()` suivant pour ajouter au `Jeu j` un marchand, un moine, un magasin et une bibliothèque, chacun alloué dynamiquement :

```
int main()
{
  Jeu j;
  // Ajoutez ici **DYNAMIQUEMENT** un moine, un marchand, un magasin et une bibliothèque :









  j.tour(); // lancement d'un tour de jeu

  return 0;
}
```

Ce programme devrait donc afficher (peut être dans un ordre différent) :

```
vend    lit    achète    écrit
```